

Aalto University  
School of Science  
Master's Programme in ICT Innovation

Dipika Baad

# **Automatic Job Skill Taxonomy Generation For Recruitment Systems**

Master's Thesis  
Espoo, May 6, 2019

Supervisor: Professor Alexander Jung, Aalto University  
Advisor: Tapio Auvinen, D.Sc. (Tech.)

Aalto University  
 School of Science  
 Master's Programme in ICT Innovation

ABSTRACT OF  
 MASTER'S THESIS

<b>Author:</b>	Dipika Baad		
<b>Title:</b>	Automatic Job Skill Taxonomy Generation For Recruitment Systems		
<b>Date:</b>	May 6, 2019	<b>Pages:</b>	70
<b>Major:</b>	Data Science	<b>Code:</b>	SCI3095
<b>Supervisor:</b>	Professor Alexander Jung		
<b>Advisor:</b>	Tapio Auvinen, D.Sc. (Tech.)		
<p>The goal of this thesis is to optimize the job recommendation systems by automatically extracting the skills from the job descriptions. With rapid development in technology, new skills are continuously required. This makes the skill tagging of the job descriptions a more difficult problem since a simple keyword match from an already generated skill list is not suitable. A way of automatically populating the skills list to improve the job search engines is needed. This thesis focuses on solving this problem with the help of natural language processing and neural networks.</p> <p>Automatic detection of skills in the unstructured job description dataset is a complex problem as it involves being robust to the ambiguity of natural language and adapting to words not seen in the historical data. This thesis solves this problem by using recurrent neural network models for capturing the context of the skill words. Based on the context captured, the new system is capable of predicting if the word in the given text is a skill or not.</p> <p>Neural network models like Long short-term memory and Bi-directional Long short-term memory are used to capture the long term dependencies in the sentence to identify skills present in the job descriptions. Various natural language processing techniques were utilized to improve the input feature quality to the model. Results obtained from using context before and after the skill words have shown the best results in identifying skills from textual data. This can be applied to capture skills data from job ads as well as it can be extended to extract the skill features from resume data to improve the job recommendation results in the future.</p>			
<b>Keywords:</b>	knowledge discovery, job recommendation systems, information retrieval, sequence labeling, context learning, entity recognition, recurrent neural network		
<b>Language:</b>	English		

# Acknowledgements

I would like to thank Professor Alexander Jung for his expert supervision and guidance throughout the thesis work. I had attended his course in machine learning which was of great help for me to conduct this thesis and follow the proper guidelines necessary for machine learning research. I want to thank VXT Research Oy for providing me the resources and giving me the opportunity for a thesis internship project in natural language processing. Especially I would like to thank my advisor Tapio Auvinen for giving me the guidance needed. Finally, I would like to thank both of them for the great support and guidance necessary to complete this project successfully.

Espoo, May 6, 2019

Dipika Baad

# Abbreviations and Acronyms

LSTM	Long Short-Term Memory
BILSTM	BI-directional Long Short-Term Memory
RNN	Recurrent Neural Network
GRU	Gated Recurrent Unit
POS	Part Of Speech
NER	Named Entity Recognition
NLP	Natural Language Processing
AI	Artificial Intelligence
FFNN	Feed Forward Neural Network
CRF	Conditional Random Fields

# Contents

<b>Abbreviations and Acronyms</b>	<b>4</b>
<b>1 Introduction</b>	<b>7</b>
1.1 Structure of the Thesis . . . . .	8
<b>2 Background</b>	<b>10</b>
2.1 Background of Recruitment Systems . . . . .	10
2.2 Background of Sequence Labeling Models . . . . .	12
2.2.1 Neural Network Architectures for Sequence Labeling .	13
2.2.2 Simple RNN Architecture . . . . .	15
2.2.3 LSTM . . . . .	16
2.2.4 BiLSTM . . . . .	17
2.3 Applications of Sequence labeling . . . . .	18
2.4 Methods for Context Capturing . . . . .	20
2.5 Domain Specific Applications of Sequence Labeling . . . . .	22
2.6 Techniques for Automatic Job Skill Detection . . . . .	22
<b>3 Problem Definition</b>	<b>24</b>
<b>4 Implementation</b>	<b>29</b>
4.1 Neural Network Architecture For Job Skill Detection . . . . .	29
4.2 Tools . . . . .	30
4.3 Data Preparation . . . . .	31
4.3.1 Data Collection . . . . .	32
4.3.2 Data Labeling . . . . .	34
4.4 Input Feature Engineering . . . . .	37
4.5 Hyperparameter Selection . . . . .	40
4.5.1 Number of Epochs . . . . .	40
4.5.2 Word Embedding Dimension . . . . .	40
4.5.3 Hidden Layer Dimension . . . . .	41
4.5.4 Mini Batch Size . . . . .	41

4.5.5	Optimizers . . . . .	41
4.5.6	Dropout . . . . .	42
<b>5</b>	<b>Evaluation</b>	<b>43</b>
5.1	Quality Metrics . . . . .	43
5.2	Results . . . . .	45
5.2.1	Hyperparameter Selection . . . . .	45
5.2.2	Comparison of Input Features and Models . . . . .	49
<b>6</b>	<b>Discussion</b>	<b>54</b>
6.1	Results Interpretations . . . . .	54
6.2	Comparison with Related Work . . . . .	56
6.3	Limitations of Experiments . . . . .	57
6.4	Future Work . . . . .	58
<b>7</b>	<b>Conclusions</b>	<b>61</b>

# Chapter 1

## Introduction

Job search engines are rapidly developing daily and a lot of research is being done in improving job matches to meet the gap between job demand and supply [52]. Major problems faced by job search engines are in finding the attributes from unstructured text data of job descriptions and resumes of candidates that improve the matches or search results. Artificial Intelligence (AI) and Knowledge Discovery have been utilized to improve these matching problems between requirements and availability of skills in the market. A survey of job recommender systems has presented the inappropriateness of the traditional approach of boolean search methods in the information retrieval methods in recruitment platforms [1]. The reason for it being inefficient is that simple keyword-based filters based on skills are not sufficient enough to consider the complexity of the person-job fit for selection decisions since they often depend on underlying criteria like personal characteristics, social skills and so on [35]. Hence, more sophisticated algorithms are being used to do the job matching which requires good feature engineering to perform well.

Rich methods extracting all the key features from the textual information in job descriptions and resumes would help new recommender approaches. Three most effective recommender approaches have been highlighted by Al-Otaibi and Ykhlef [1] are Content-based filtering, Collaborative filtering, and Knowledge-based approaches. Content-based filtering and Knowledge-based recommender systems need good domain-specific feature engineering requirements. This shows the need to have an automated system that can generate domain-specific features from unstructured data.

Many of the job portals are using matching algorithms to find the match between a job seeker and job opening faster. If most important features like skills can be extracted from the text automatically then this would potentially speed up the feature generation process. This skill detection technique can be extended to get more features from a text other than skills as well.

This thesis focuses on solving this issue of key feature extraction to enrich the knowledge base of skills with automatically updating the new skills as they are found out in job descriptions. With the rapid pace of technology development, the job market is becoming more dynamic. Skills expected on the job are continuously changing which requires the job portals and e-recruitment platforms to become more robust in handling these new skills and automate the skill list creation. Extracting skills to be used as input features to job matching algorithms would require frequent updating in this dynamic environment. Thus an automatic way of extracting the skills from the job descriptions is needed. A simple keyword-based matching using old skills taxonomy would not work when new skills are going to appear in the job descriptions in the future.

A smart way to generate the skills taxonomy based on the job descriptions and resumes uploaded on the job search platforms is needed. The language used in the job descriptions is quite standard and not artistic or varying in nature as compared to the blogs or articles, this opens a door for using the natural language processing to learn the skills occurrences based on the context in which it appears in the text. The solution proposed in this thesis is to use various context learning techniques to learn the context for the job skills and generate the skill set which can be used for auto-tagging the applications for generating better matching results. Job skills will appear in the descriptions quite often where they are being expressed as needed or what the company is doing. Those will be followed by a similar structure even though the choice of words is made differently in different descriptions. Hence, learning these patterns to identify the skill words is proposed to automate the task of skill feature extraction. Various neural network techniques that were used for sequence learning have been used to solve similar problems. Long short-term memory (LSTM) [22] and Bi-directional long short-term memory (BiLSTM) neural network techniques that learn the long term dependencies in the sequences are used to detect skill words in the job description data.

## 1.1 Structure of the Thesis

This thesis initially describes relevant literature in Chapter 2 regarding the similar problems related to keyword extraction and various techniques implemented which have proven to work for context-based keyword detection. This chapter provides a background to techniques implemented. The literature review also covers research conducted in other areas where the problem is similar to context learning and sequence labeling. Chapter 3 defines the problem in terms of machine learning problem and how machine learning



can help to solve this problem. Following Chapter 4 covers techniques implemented in this thesis to solve the problem of skill detection. It describes the custom model built for the problem being solved. The other sections in this chapter describe the initial data collection processes and various data cleaning techniques implemented to set the environment right for implementations. Experiments done with various parameters and comparison of methods are described in the next Chapter 5. Finally, evaluation metrics and visualization of results are illustrated with charts and tables in the Chapter 5. Chapter 6 includes discussion about the results obtained and limitations of the application in real-world applications and the future scope of the experiments. Chapter 7 explains and concludes the thesis with outcomes found out from the experiments.

## Chapter 2

# Background

### 2.1 Background of Recruitment Systems

An automated system for intelligent screening of candidates for recruitment using ontology mapping is described by Kumaran and Sankar [29]. The system uses keywords and an ontology for automating the task of matching the jobs to resumes. Ontology documents are constructed for the features of candidates and job openings are also represented using an ontology. Finally, these ontologies are mapped to retrieve the relevant candidates for the given job description. This is an advanced technique proposed over the currently restricted Boolean Keyword Search method. They have used “Concept Linking” which is a process of connecting related documents by using commonly shared concepts. Underlying data models used in their study were ontologies which are in the form of concepts, properties, relations between concepts, relations between properties, axioms, instances of concepts and properties. Ontology mapping seeks to find the mapping of elements of one ontology over another which have the same intended meaning. Ontology is generated for features like personal information, employment, skills, interests, etc. To extract this information they have used concept extractor that uses named entity recognition. Their method has shown to give 90% accuracy on job matching. This shows the importance of using feature extraction from the job texts to improve job matching. Automated efficient of skill extraction is one of the necessities for such search engines.

Laumer and Eckhardt [32] have used the technique of collaborative filtering combined with the content-based matching approach to recommend relevant jobs to candidates. Collaborative filtering is a technique in recommender systems to predict the interests of the user based on the collective preferences of many users. The content-based approach involves using differ-

ent features of the matching entities. Features could be attributes of matching entities for example for product recommendation the product attributes like color, model, etc. and for persons searching products, it would be their personal preferences retrieved from profile description. Content-based recommendation systems work efficiently when the items to be recommended are represented in the structured format rather than unstructured format [49]. Before an estimation of the user requirements is made using some mathematical models or algorithms, the input must be converted into a format that is easy to process for algorithms [49]. Thus representation of the entities is a crucial step in content-based recommendation systems. Using structured data as input would have a small number of characteristics that can be used as an n-dimensional input vector. This is more efficient since the input is restricted to a limited number of characteristics with well-defined values [49]. On the other hand, when unstructured data is used as input, this involves a lot of ambiguity such as the same feature that can be depicted in multiple ways. For recommendation system, only the relevant information necessary for making a recommendation is of high importance. Understanding the natural language is still a complex problem that even AI systems cannot fully grasp it until now [49]. Thus, having structured data extracted from the textual data is necessary for building efficient recommendation systems. This is where the need for automatic feature extraction can play a crucial role.

Another study of finding experts based on the user profile abstracts is mentioned in [62]. This is one of the key topics in information retrieval. In this paper, the process of spreading is used which finds the related terms using the ontology in WordNet<sup>1</sup> or Wikipedia<sup>2</sup> and then uses it for enabling user profile matching. One of the drawbacks of traditional content matching algorithms for profile matching mentioned is the use of bag of words (BOW) representation of the profiles and job descriptions to find the relevant matches which do not take into consideration the semantics in its representation [62]. Semantic relationships which are not explicit in the text gets ignored. They extended the current BOW representation by including additional related terms by referring to an ontology. Input to the matching system involves giving weights to the terms as well. This is based on the user profile extraction which is a set of topics pertaining to the user and then extending the system to find relevant ontologies for improved matching. This again shows the importance of key information extraction from the descriptions as using direct content matching is not a very efficient way for recommendation systems. Their findings show that more sophisticated approaches capturing the

---

<sup>1</sup><https://www.w3.org/2006/03/wn/wn20/>

<sup>2</sup><https://wiki.dbpedia.org/services-resources/ontology>

semantics of the text are needed for improved matching and simple content matching is not efficient.

## 2.2 Background of Sequence Labeling Models

Deep learning which is one of the currently used machine learning techniques have achieved great success in the area of image analysis, speech recognition and language understanding [73]. Deep learning uses supervised and unsupervised techniques to learn the multi-level representations and features for problems like classification and pattern recognition [73]. This family of learning models includes convolutional neural network (CNN) [71], Recurrent Neural Network (RNN) [17], autoencoders, etc. which are often used in text understanding models. Sequence labeling tasks involve understanding the sequence patterns in the text data and utilizing those features for labeling the sequence of inputs to output labels. The research has shown the applicability of deep learning models in text learning and sequence labeling and hence these methods have been explored in this thesis.

Collobert et. al. [11] proposed a simple feed-forward neural network (FFNN) for sequence labeling tasks which requires little feature engineering and can learn from trained word vectors from a huge corpus. The limitation of that method is that it can only consider a fixed sized window around each word for prediction. This approach discards useful long-distance relations between words [6].

A well-studied solution for the neural network model to be able to consider variable length input and have a long term memory is the RNN. RNNs have shown great success in various sequence learning tasks such as speech recognition [18], machine translation [8], and language modeling [42]. LSTM is a form of RNN with a nontrivial recurrent unit containing forget gate which allows it to retain information from a long distance. BILSTM model can take into account the arbitrary length of context on both sides of a word which is an advantage over the limited context in FFNN. LSTM and BILSTM are used in this thesis for capturing the context to predict the skills present in the job texts.

Part of speech (POS) tagging where the word is assigned a tag that represents its part of speech based on the relation to its surrounding words is one of the sequence labeling tasks in NLP. Using BILSTM for tagging the POS has been shown to give results comparable to state-of-the-art POS tagging solutions [66]. Named entity recognition (NER) which is one of the chal-

lenging tasks in NLP has also been proven to work well using BiLSTM. This traditionally required large amounts of knowledge consisting of lexicon and in the form of feature engineering. Using word and character level features with BiLSTM, it has been shown to give results competitive on the CoNLL-2003 dataset which is largely used for comparing the NER performance [7]. The performance of these RNN models on sequence tagging tasks has shown the applicability for many other sequence tagging tasks as well. In this thesis, we use these models for tagging where our problem is similar to finding the tags which indicate skill or non-skill in a sequence of words. In the following sections, we will see how these neural net models are able to capture the context and make predictions based on the context before.

### 2.2.1 Neural Network Architectures for Sequence Labeling

Neural network models chosen for job skill detection are such that they can remember the past inputs. In order for a network to learn the sequence of words, there needs to be a mechanism present in the model which enables it to memorize the words preceding that. RNNs are network models that provide this mechanism over the normal feed-forward neural network. RNNs can remember the past hidden state which essentially helps to steer the final output which is the tag of the word.

The Recurrent Neural Networks (RNNs) family is considered to solve this problem as they work on sequential data [31]. RNNs take sequence  $X = (x_1, x_2, x_3, \dots, x_n)$  input and return output sequence labels  $Y = (y_1, y_2, y_3, \dots, y_n)$  that depicts the sequence information at each step of input. Theoretically, they can learn from long sequences but in practice they fail to capture long dependencies. Thus LSTM models have been utilized in applications where long term dependency of information needs to be captured. Long term dependencies are essential to capture in NLP tasks since words distant from the current input word can be indicative of what is going to follow. In job skill detection the words indicating the need versus company descriptions are dependent on the words far from the skill words. For example, in job description like “You will have the opportunity to leverage our massive datasets, work with the latest web technologies like d3 and react” has words *You will have* way before the *technologies* which shows that it is the requirement. In contrast, if those words were “We have been working with massive datasets where technologies like d3 and react are being used”. This would make those skills as non required skills and the sentence being part of the company description. Short content of “technologies like” is not

sufficient when one wants to extract specific attributes from the large texts having multiple occurrences with a different contexts in the same text. When such ambiguities exist then capturing long term dependencies using LSTMs would help to use the long history for the prediction model.

LSTMs capture the sequence before the current input for prediction of the current input and if one wants to consider the sequence information that follows the current word then BILSTMs are useful. Performance of LSTM versus BILSTM wholly depends on the text structure if the words followed by or preceding are more helpful in predictions. To improve the prediction using LSTM and BILSTM the type of input feature also has an influence on the prediction quality.

The words fed to a neural network are often represented as word embeddings [39] which are learned from a corpus [34]. Word embeddings are distributed representations of words which are real-valued and low-dimensional vectors. Each dimension is representative of syntactic or semantic properties of the word [34]. They essentially make the word representations of similar words to have similar representation such that word vectors of similar meanings are closer to each other in the vector space. This usually has fewer dimensions around tens and hundreds. The distributed representation of a word is learned based on the usage of words in the corpus. This ultimately allows words that are used in similar contexts to have similar representations. Word embeddings learned using the Word2Vec model have given state-of-the-art word embeddings [16]. Research done in sequence tagging involving non-domain specific tasks like POS tagging, NER, chunking etc. have used pre-trained word embeddings from Google<sup>3</sup> trained on 100 billion words from Google News, Senna embeddings<sup>4</sup> trained on Wikipedia and Reuters RCV-1 corpus, Glove<sup>5</sup> trained on Wikipedia and web text. For domain-specific tasks, word embeddings learned from the data available for training are used as input [51].

Sequence tagging studies have used the information other than just the word representations [6] as input to the network. Collobert et. al. [11] have used capitalization features like: allCaps, upperInitial, lowercase, mixed-Caps, and other. Other information like suffix information, POS tag information, etc. are also being used to improve the performance of the tagging.

---

<sup>3</sup><https://code.google.com/archive/p/Word2Vec/>

<sup>4</sup><http://ronan.collobert.com/senna/>

<sup>5</sup><http://nlp.stanford.edu/projects/glove/>

### 2.2.2 Simple RNN Architecture

RNNs are distinguished from the feedforward neural networks by having a feedback loop in them. The loops are used to feed the previous state for using the information from the past. In simple RNN architecture, the recurrent units are typically simple tanh or Rectified Linear Unit (ReLU) functions. More sophisticated recurrent hidden units such as LSTM are explained in the following sections. Ultimately this facilitates the flow of information from the previous state while going through the next input. The input will be provided with one word at a time so the input at time  $x_t$  will receive the  $h_{t-1}$  the previous hidden state when  $x_{t-1}$  was fed to the network. In Figure 2.1, the  $W^{hh}$  is the additional weight matrix which will help to learn the weights for connecting the previous hidden state to the next. The network on the left side is the RNN shown without unfolding and on the right side unfolded architecture is shown to better understand the information going through the network. It looks similar to a multi-layer feed forward network with the only exception that the  $W^{hh}$  is the same across the layers. Input  $x$  dimension is of equal size to the size of the word representation used in the task. Essentially, now the output is not just a function of the input but is the function of the current input and previous state. Input is only used to change the state of the memory stored, unlike the usual network where it has a direct effect on the output.

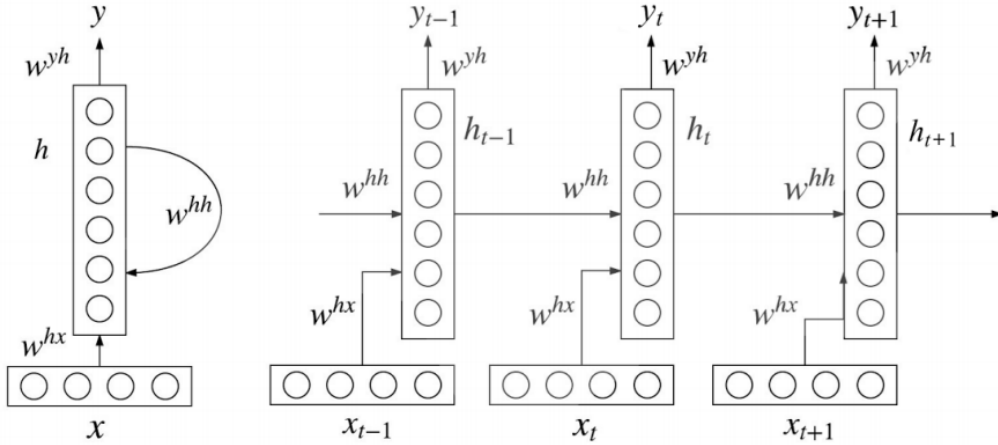


Figure 2.1: RNN Architecture [72]

At each step, the output  $h_{t-1}$  generated from the previous input  $x_{t-1}$  is fed to the processing and this is followed until the last element in the sequence. The equation for calculating the  $h_t$  at time step  $t$  is as follows:

$$h_t = f((W_{hh})(h_{t-1}) + (W_{hx})(x_t))$$

Here the  $f$  is usually the tanh or ReLU function. The output  $y_t$  at time step  $t$  will be the probability distribution over the classes which can be derived from the softmax function.

$$y_t = \text{softmax}((W_{yh})(h_t))$$

For the network built for solving this problem, we have used negative likelihood loss which takes log probabilities and hence logsoftmax function is used for the experiment, which gives the values for two classes. The highest value bearing class is used as an output label for the  $x_t$  input.

### 2.2.3 LSTM

LSTM is a special type of RNN with more sophisticated hidden unit than simple RNNs which overcomes the vanishing gradient problem of RNNs which is the cause for simple RNNs inability to remember the long contexts. As the information flows from input to output, the error flows from output to input through different time steps in the case of RNNs. Here different time steps can be considered as different layers in the neural network when looking at the unfolded architecture of RNNs. Gradients diminish in the value as they go further from output towards the input layer. This leads to less weight updating for the previous layers and ends up not learning much from the previous inputs. This is termed as vanishing gradient problem. LSTM and Gated Recurrent unit (GRU) [15] are used to tackle this issue by using complex units that enable the network to remember the long term contexts. In LSTM, there are four interacting neural network layers instead of one and in addition to the hidden state they also have cell state. The architecture of the LSTM is as shown in Figure 2.2.

LSTM makes the decision of what information to drop first which is done in the sigmoid layer called “forget gate” in the cell state. The sigmoid function takes  $h_{t-1}$  and  $x_t$  (current input) and outputs the number between 0 and 1 where 0 means forget and 1 means completely keep it.

$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$

The next decision is what information to store in the cell state which consists of two steps. The first step is to know which values need to be updated that is done by another sigmoid function/layer called as an input gate. Then the tanh function/layer chooses the candidate values  $\tilde{C}_t$  which needs to be added to the cell state.

$$i_t = \sigma(W^i x_t + U^i h_{t-1})$$



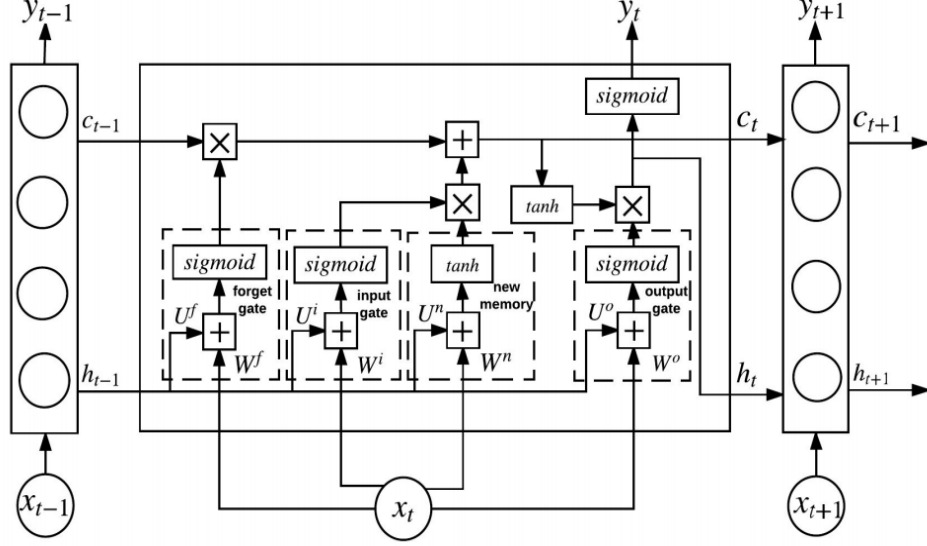


Figure 2.2: LSTM Architecture for Sequence Tagging [72]

$$\tilde{C}_t = \tanh(W^n x_t + U^n h_{t-1})$$

The new cell state is obtained using these as follows:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

In the end, the LSTM decides the output based on the cell state. It initially runs a sigmoid layer which makes the decision regarding which parts of the cell state goes to output called as an output gate.

$$o_t = \sigma(W^o x_t + U^o h_{t-1})$$

LSTM gives the cell state to the tanh function and multiplies that with the output of the sigmoid gate so LSTM ends giving the output as given in the equation below.

$$h_t = o_t * \tanh(C_t)$$

#### 2.2.4 BILSTM

BILSTM has two LSTM networks together. The input sequence is fed normally in time order to one network, and in reverse to the other network. The outputs of the two networks are concatenated at each time to get the output

for the corresponding input. The advantage of this is on top of learning the previous context the network can learn the context after the current input as well to decide the output for the current word. Basic architecture for sequence tagging using BILSTM is as shown in Figure 2.3. LSTM units as depicted in the BILSTM architecture have the same architecture as shown in 2.2. The architecture shows how the backward and forward context is used for predicting the label at any time step  $t$ .

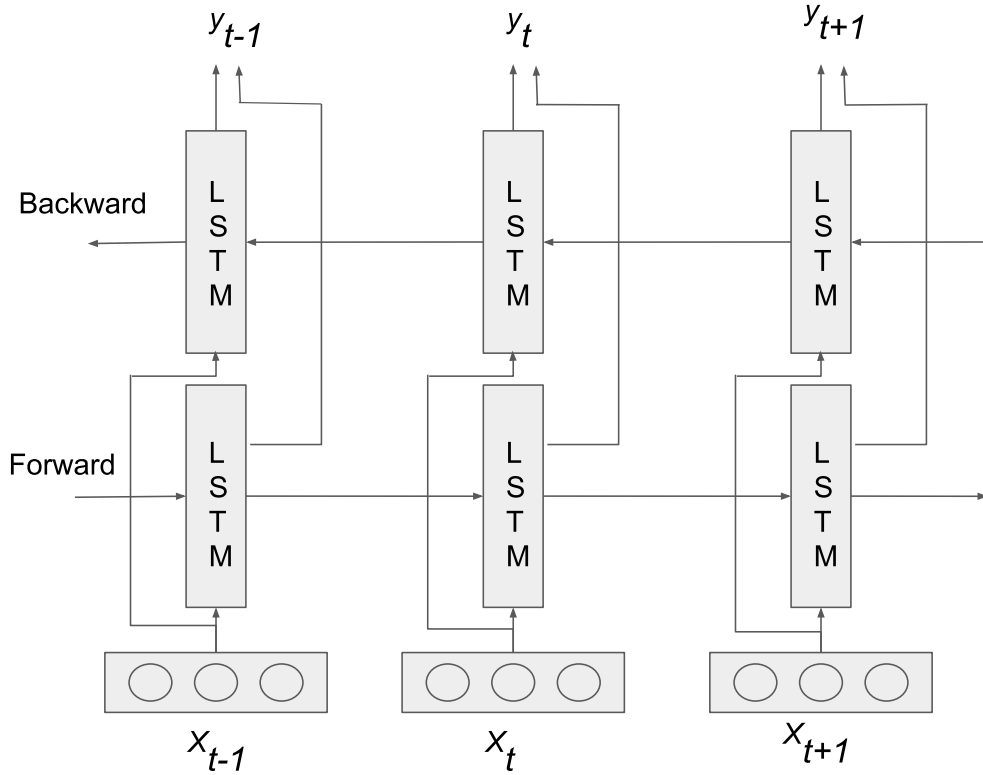


Figure 2.3: BILSTM Architecture for Sequence Tagging

## 2.3 Applications of Sequence labeling

Keyword extraction from the text can be formulated as a sequence labeling problem where keywords are identified based on the context in which it exists. Understanding the context essentially means capturing patterns from the sequence of words. Context memorizing task is achievable with the use of recurrent neural networks RNNs as mentioned before. Research of these neural net models in applications involving context learning and comparisons

of those to traditional methods like n-grams has been done and is an ongoing area of research. One such study in the application of RNNs for language understanding is presented in [41]. Statistical language modeling consists of predicting the next word in textual data given context which is a sequential data prediction problem. Mikolov et. al. [41] have shown reduction of perplexity by 50% in comparison to state-of-the-art backoff language models like machine translation [9] or Optical Character Recognition (OCR). Perplexity is a way to measure how well a probability model predicts a test set, in case of NLP tasks it is used for evaluating language models. They used simple RNN architecture for language modeling with application in speech recognition.

An artificial neural network has been proposed by Bengio [2], who used feed-forward neural network with fixed length context. The major drawback of this method is that it uses the fixed length context which needs to be specified before training [41]. This would restrict the model to see only five to ten words before the word when predicting the next word. RNNs do not use fixed length context and therefore information can cycle inside the network arbitrarily for a long time. Another advantage mentioned in [41] is that using the RNNs require only setting the parameter of hidden size layer whereas feedforward NN requires a lot of parameters.

RNNs used in slot filling [59] tasks also have shown to work well compared to traditional methods for spoken language understanding [38]. Slot filling is similar to named entity recognition (NER) with the only difference being the information being searched for is with respect to some other words. For instance, if two location names are mentioned in the sentence describing the train journey then instead of tagging city names as location entity, they would be assigned concepts like “arrival city” and destination city based on the sentence structure. Thus, the relation between the entity and other words has to be captured for a slot filling task. One major application of slot filling is to have better human-machine dialogue. Another usage of slot filling is in knowledge base population tasks<sup>6</sup>. Slot filling task is treated as a sequence classification task which is similar to the problem being solved in this thesis. One example of a slot/concept annotation problem is identifying in/out/begin (IOB) representation in which for example trying to find the arrival and destination cities mentioned in the sentence. Slot filling takes sequence of words and tags them with slot/concepts (arrival city/departure city in the above example) [38]. Mesnil et. al. [38] used word embeddings as input instead of n-gram models. N-gram is a vector encoding that takes sequences of tokens of fixed length N into account. Each n-gram is treated as

---

<sup>6</sup><https://nlp.stanford.edu/projects/kbp/>

a normal word [75]. N-gram models suffer from data sparsity as the number of all possible n-grams will be  $V^N$  given the size of the vocabulary is  $V$ . Word embeddings show good generalization properties across various NLP tasks [12].

## 2.4 Methods for Context Capturing

Context capturing in neural networks can be done with the help of recurrent neural networks which have the mechanism of memorizing the previous inputs. Different techniques of implementing memorizing units exist as mentioned before. LSTM are used to capture long term dependencies. LSTM networks have been employed to many natural language processing tasks involving sequence tagging like POS tagging [66], chunking [6], NER tagging [6], Punctuation Restoration [64] [65] [68], Sentence Boundary Detection [25], Spoken Language Understanding [70] and so forth. RNNs have been used for parsing [56], sentiment analysis [57], paraphrase detection [55], question answering [24] and logical inference [4]. All of these tasks which require sequence labeling have used some form of RNNs to capture the context in order to use context as input to predicting the label for a given word.

Even applications using automatic speech recognition usually use RNNs over n-gram models for various tasks. One of these studies in speech recognition is in the first pass decoding where they improved the rare word prediction with approximated RNN based language models [53]. First pass decoding is the pass where recorded speech is converted into words with the help of acoustic model, lexicon and language model [30]. In their experiments on large English-based corpora, these approximated RNN language models have shown benefits over conventional N-gram models [1, 3, 2, 4] in first-pass decoding. RNNs have been used in the language models for first pass decoding. RNNs used are not only for sentence level understanding but also used to capture the word representations by using character level characteristics [67]. In their research for understanding the internal structure of the word [67], using pre-trained word embeddings have shown to give significant performance improvements.

Various LSTM network architectures have been evaluated for a variety of sequence labeling tasks in NLP. It has drawn a lot of attention in research because its output is useful as an input to many downstream applications [23]. Huang et al. [23] compared various neural net models like LSTM networks and BILSTM. LSTM networks with a Conditional Random Fields (CRF) layer (LSTM-CRF), and BILSTM networks with a CRF layer (BILSTM-CRF) for sequence labeling task. The results were compared to the NLP

benchmark sequence tagging data sets. Results showed that BILSTM CRF is more robust and has less dependence on word embeddings [23] as compared to the observation in [12]. The application example presented was in product ads recommendation where names can be extracted from the user query. Entities can be extracted from the product description as well. These entities extracted from both the query and description can be used to identify spans of the text containing the user information and attributes retrieved from the product description which would ultimately help in retrieving relevant product ads [23]. The tag information extracted from the description is used in improving the product search engine by retrieving the most relevant results.

Although RNNs have been applied to produce promising results on a variety of tasks including language model [41] and speech recognition [19], they cannot remember the long-range dependencies of input sequences. LSTMs, on the other hand, are similar to RNNs but the hidden layer updates are replaced by purpose-built memory cells which helps in remembering the long-range context. CRF networks on the top help to utilize the tagging information of the neighboring elements for tagging the current input unit. Huang et al. [23] have used 130k vocabulary for training the word embeddings. Results were that BILSTM and BILSTM CRF models have resulted in better tagging accuracy compared to others with accuracy around 97% [23]. Difference between these two is not huge, so depending on whether one can utilize the tag information in the output tags from neighboring elements would affect the choice of the LSTM model. For Job Skill detection, the output tags are just two (skill / non-skill) and hence, using the CRF layer would not help the model as such.

Another study in comparing the different LSTM models for the POS tagging, chunking, and NER tagging tasks have shown similar results with BILSTM network performing better when compared with state-of-the-art performance on these tasks [6]. Models that incorporate BILSTM with character level features extracted using CNN have shown results competitive with the state-of-the-art in POS tagging, chunking, and NER [6]. They also showed that these work better even without using any hand-crafted features or external knowledge which makes them robust and easily applicable to other domains.

## 2.5 Domain Specific Applications of Sequence Labeling

Sequence labeling apart from usual NLP tasks have been utilized for domain-specific applications to label text for retrieving attributes needed for improving domain specific recommendation systems. Bidirectional RNNs have been used in the event extraction and attribute retrieval from biomedical Electronic Health Records (EHR) which has helped to understand the semantics of the EHRs [25]. Another relevant application of attribute finding from the domain-specific text was done for product discovery in E-commerce listings. The authors used complex neural net models such as BILSTM for sequence labeling for extracting the features from product listings [51]. Labeling the attributes like color, fabric, brand, shape, etc. and associating these attributes to relevant items to form meaningful products has improved performance on recommendation and searching of products. They used BILSTM for keyword labeling which gave around 98% accuracy for prediction. They evaluated the accuracy separately for in-vocab labeling and out-of-vocab labeling which shows insignificant differences between these two categories as well. In-vocab list contains the keywords present in the training and validation dataset and the out-of-vocab list refers to the keywords not present in the training and validation dataset. A similar application of sequence labeling has been done in chemical NER [21]. They used BILSTM with the CRF layer for locating chemical named entities in the literature which is an essential step in chemical text mining tasks for recognizing chemical mentions, properties and relations amongst them. Their model achieved an F1 score of 90.04.

## 2.6 Techniques for Automatic Job Skill Detection

Research in the area of job skill detection from resumes and job descriptions is ongoing to automate and improve the recruitment systems. One such study in information extraction from resumes was conducted where English and Japanese datasets were compared [28]. Methods implemented for these were BISLTM-LSTM-CRF where character level embeddings were learned using LSTM layer [28]. The experiments in their research were conducted for prediction using different settings like 1. without character embeddings or word embeddings, 2. with character embeddings but without word embeddings, 3. with pre-trained word embeddings, without character embeddings,

and 4. Character level embeddings with pre-trained embeddings. The highest F1 score obtained was 76.07 for English resumes with pre-trained word embeddings and character embeddings with BILSTM-LSTM-CRF [28].

Research conducted by Zhao et al. in skill identification and normalization for the CareerBuilder<sup>7</sup> online job recruitment service showed accuracy of 82% and recall of 72% in skill tagging task with Word2Vec method for tagging the skills [74]. This accuracy was obtained with 100M input data [74]. Word2Vec vectors trained on the training dataset were used for detecting the skill words by finding the similarity between the skill word vectors and the word vectors of the word under consideration.

After researching the different studies done in the area of sequence labeling and keyword extraction tasks, most promising methods like BILSTM and LSTM have been found to give good results and are widely used in complex NLP tasks currently. Different ways of providing input to the network including character level and word level embeddings have been shown to give good results in these studies. These neural net models have been selected due to the robustness of adapting to new data as well as memorizing capability for the varying lengths of context. All these factors suggest these methods would help the skill detection task and help build a system to detect new skills automatically with less feature engineering efforts.

---

<sup>7</sup><http://www.careerbuilder.com/>

## Chapter 3

# Problem Definition

Current systems using just the job titles for finding the relevant candidates or job seekers finding job openings based on job titles is clearly not an efficient way of matching jobs. As complex jobs may require a lot of skills, job titles are not necessarily representative enough to improve job search engines. A person having one skill might be needed in many job roles involving that skill and job recruiters looking for a position might have the same issue when looking for candidates who have similar skills but with different titles. This ambiguity of expressing job titles as well as jobs involving multiple skills needs a search engine that could potentially extract the features from job descriptions and resumes automatically to improve the match rate essentially. Using the complete job description text or resume text for matching is not sufficient since many misleading words that are not indicative of the skills would result in misleading matches in the system. Traditional job matching algorithms rely on creating an exhaustive skill taxonomy manually or using some existing skill taxonomy which needs to be updated either manually or by keeping track of new technology advancements. An automatic system that could update this skill list based on new job descriptions is needed.

Due to the ambiguity of natural language used for job descriptions and different styles of description articulations, simple rule-based techniques cannot be used to capture the skills from these textual descriptions. Even if it works to some extent, it is not scalable to other similar problems of essential keyword extraction problems. Hence, a more generic solution that could learn the context automatically on the historical data to be able to extract new skills is needed. This will not only save the time of generating skills but could be applied to other applications requiring the recommendation/search based on textual descriptions.

In this thesis, the above problem is solved using neural networks. Deep neural networks have been used in the areas where they can potentially in-



fluence the current processes by automating the tasks which require human cognition but at the same time are monotonous [14]. Clearly going through descriptions and finding the skill is a repetitive and monotonous task. It is also not straightforward enough to be solved by rule-based systems. Hence, this is an apt problem for AI. As seen in the previous sections, research in text processing and information retrieval have been empowered with the use of deep neural networks. The following sections will explain how we transform this business problem in a way that allows using AI for tackling the manual task of skill detection.

AI systems are highly dependent on the nature and quality of input similar to machine learning problems. Making the system more reliable, various methods for improving the performance of the model trained needs to be analyzed. First, we define various components used in the system. The definition of the problem is based on guidelines given in [27].

Data is characterized by features and labels. For job skill detection, textual data is the raw input. Job description data in text format cannot be used as input directly to the system. It needs to be transformed into some numerical representation for the model to learn patterns. There are multiple ways of representing the textual data in numerical format. Bag-of-words method [45] which does not take into consideration the order of words and is simply the frequency of the words in the input. Topic modeling techniques like latent Dirichlet allocation (LDA) [45] or Latent Semantic Analysis (LSA) which capture the global context have been used in language modeling [46]. This can be used as an input vector for representing the text. Commonly used representation for sequence labeling problem in neural networks is word embeddings learned using Word2Vec model. Word embeddings learned from the corpus of all job descriptions are used in this thesis. Word2Vec [44] essentially is a dense vector capturing the word context as well. As our goal is to use the context for predicting if the word is a skill or not, using this vector as input feature which is representative of the semantics of words is essentially feeding better features to the network compared to just bag of words. Word embeddings are a better representation over a bag of words because the dimensionality is low comparatively.

Each data point is a sentence of variable length of words. Each word is represented as a continuous dense vector learned from the Word2Vec model. Let  $W$  be the 2D-array (two dimensional) of word vectors of word embedding dimension of  $D$  and number of rows ( $v$ ) equal to unique words in the job description dataset. Then,

$$W = (w_1, \dots, w_v)$$

$w_i$  is the  $i^{th}$  word vector of dimension  $D$  where  $i$  is the id of the word.

Once these vectors are created the datapoint in the dataset is the vector of concatenated vectors of each word in the sentence(datapoint)  $x_j$  where  $j \in 1, 2, \dots, n$ .  $n$  is the total number of input data points. Let the number of words in the sentence at  $j$  datapoint be  $m_j$  and let  $k^j$  be the array of word indexes in the sentence. The lengths of the sentences are different. Input at datapoint  $j$  will be of dimension  $m_j \times D$  and will be represented as follows:

$$x^j = (w_{k_1^j}, w_{k_2^j}, \dots, w_{k_{m_j}^j})$$

For instance, if sentence at location 4 is “python expert needed” and assuming the word vectors for words are as follows for embedding size of 4:

$$python = (0.05, 0.07, 0.09, 0.3)$$

$$expert = (0.04, 0.03, 0.07, 0.4)$$

$$needed = (0.02, 0.23, 0.03, 0.2)$$

Then, the input at  $x^4$  would be as follows in 3.1:

$$\begin{aligned} x^4 = & ((0.05, 0.07, 0.09, 0.3), \\ & (0.04, 0.03, 0.07, 0.4), \\ & (0.02, 0.23, 0.03, 0.2)) \end{aligned} \tag{3.1}$$

This is of size  $3 \times 4$  where 3 is the length of the words in a sentence and 4 is the embedding dimension. The corresponding output vector would be of size  $m_j$  as well with  $m_j \in \{0, 1\}$  as the label for each word will indicate if the word is a skill or not. The network is given one word at a time and the corresponding label is predicted thus the feature mapping is from sequence of word vectors to sequence of the same size having 0,1 values. Feature space is any real value of sentence.length  $\times$  D and label space is  $\{0, 1\}$  of sentence.length size. Label for input  $x^j$  is a vector of length  $m_j$  with each value being 1 or 0. Label 1 means the word is a skill. It is a non-skill word if the label value is 0. For instance, the correct label output for the above example would be:

$$y^4 = (1, 0, 0)$$

This is a classification problem, with each word classified as 0 or 1. Label for word is predicted on the basis of the context (surrounding words). If LSTM is used, then all the preceding word vectors' influence is considered

for predicting the label of the current word and if the BILSTM is used then the word vectors of words preceding and following the words will be input to the hypothesis function for predicting the label of the current word. Details of function are explained in Chapter 5. The function will take the word vector of the current vector and modified the hidden state until the current word for predicting the label.

Features in addition to the word embedding vector added are part of the speech tag and capitalization indication of the first letter. These additional features are added to boost the accuracy of the model as they are providing features that are providing characteristics more specific to skills. When these features are used in addition then the input vector of the word is  $(D + 2)$  and when only one of these features is added then the input vector of an individual word becomes  $(D + 1)$ . This is the only difference when additional features are added. The output label size remains as is. For a neural network, the input at any given instant is the word input vector and output is the prediction label 1 or 0.

For the deep neural network used in this problem, the input is given to the network one word at a time and the input is a vector of real values of size  $D$  in case of only word embeddings are used,  $D + 1$  if the Embedding and one of the POS or letter case indication is used otherwise it is  $D + 2$  if all the features are used. The output nodes are two, one for each class. The output node gives probabilities between the range  $[0, 1]$  for each class. A label is given to the word based on the max probability of the label node amongst the two. As negative log-likelihood loss function is useful training the classifier with 'C' classes ( $C$  = number of classes), it is being chosen for this task<sup>1</sup>. This requires the log probabilities of each class as input and hence `log_softmax` function is used on the output layer to get the tag log probability values for each class whose values could range from  $[-\infty, 0]$ . The hypothesis space is as follows when only word embedding is used as input and two classes are predicted ( $C=2$ ):

$$h: \mathbb{R}^D \rightarrow \mathbb{R}^C$$

Activation function used in LSTM and BILSTM is `tanh` by default which is non-linear activation leading to an extremely large class of predictor maps [27].

Training of the neural network involves learning the weights and biases of the network during the training process. It starts with random values, the values get adjusted over and over again till they have reached peak performance. This updating is possible with first calculating the error of prediction.

---

<sup>1</sup>[https://pytorch.org/docs/stable/\\_modules/torch/nn/modules/loss.html](https://pytorch.org/docs/stable/_modules/torch/nn/modules/loss.html)

Error is the difference between the predicted values and the expected values. The goal of the neural training is to optimize and minimize the cost. This is achieved through the backpropagation mechanism. For LSTM and BILSTM this happens through backpropagation through time as there are recurring layers in the network.

The negative log-likelihood loss function is used as a loss function for where negative log-likelihood of the ground truth label sequence  $y$  for a given sentence  $x$ . It has been used for sequence labeling tasks [26][69]. The labeling cost for the sentence is the sum of cost per word at time step  $t$  (words are given in sequence) for RNN models. Any optimizers like stochastic gradient descent, Adam, AdaDelta, etc. then can use the loss function as shown in equation 3.2 below to optimize the model.

$$L(x, \hat{y}; \theta) = - \sum_t \sum_{y_t} \delta(y_t = \hat{y}_t) \log P(y_t | x; \theta) \quad (3.2)$$

## Chapter 4

# Implementation

### 4.1 Neural Network Architecture For Job Skill Detection

For job skill detection task, we have implemented the LSTM and BILSTM neural network models. As mentioned in previous Chapter 2, these RNN models help to memorize the context which can be used for predicting the label for the word under consideration. Here, we describe how the custom models are built for sequence tagging with different features given as input to the model for job skill detection task. In job descriptions, a sentence from the job ad is considered as one training instance. When the sentence is fed for prediction, the neural network is fed with one word at a time in sequence. The output is the corresponding label for each word in a sentence. The label indicates if the word is a skill or not. Label 1 indicates it is a skill and label 0 indicates that it is not a skill. In addition to the word vector, POS information and capitalization information of the first letter of a word are included. With POS information the example input and output are shown in Table 4.1. As the POS tag string cannot be given as input directly, the unique integer is assigned for each tag and then that integer id is given as input. For capitalization information, if the word has the first letter capitalized then 1 is fed for that particular word otherwise 0.

Table 4.1: Illustration of Input information and Output vector to NN

<b>Word Input</b>	We	are	looking	for	strong	skills	in	Python
<b>POS Input</b>	PRP	VBP	VBG	IN	JJ	NNS	IN	NNP
<b>Output</b>	0	0	0	0	0	0	0	1

Input to the network is the sequence of words. Each word is represented as

word embeddings. Word embeddings are a vector representation of the word in vocabulary in a low dimensional space that captures the context, semantics and syntactic similarity in relation to other words. Word embedding is a technique for language modeling and feature learning, which turns words in vocabulary to vectors of continuous real numbers (e.g., java = (... , 0.15, ..., 0.23, ..., 0.41, ...)). Each dimension of the embedding vector represents a latent feature of a word. The vectors may encode linguistic regularities and patterns. POS tag for word and the capitalization of the first letter of the word are also added as an input to the final network.

Word embeddings of the word were generated from the Word2Vec model [43] and various dimensions have experimented to see which one works better for job skill detection task. LSTM and BiLSTM implementations are using these input features of words as input and output the predicted label. Architectures for these are as shown in Figure 4.2 and Figure 4.3. The architecture shown is for an example input of “We are looking for Python Expert” to the network. The number of input neurons will depend on the embedding size and extra features used in the input like POS and capitalization information. The expected output vector would label *Python* as 1 and the rest of the words as 0. Two neurons in the output layer predict the probability of given word being label 1 or 0. Label for a particular word is label neuron which has the maximum probability. In case of LSTM, words [“We”, “are”, “looking”, “for”] will influence the prediction of the skill word *Python*. In BiLSTM case, all the words before and after the word *Python* are influencing the prediction of the label for word *Python*. Complete architecture with all the features used as input to the network for BiLSTM is as shown in Figure 4.1. The complete architecture depicts how the features are concatenated into one single vector and fed as input to the network as well as how the outputs from backward and forward LSTM units are concatenated to produce the final predictions. Architecture is shown for example input of “Python Expert Needed” as input sentence where the output will be [1, 0, 0] labeling the skill *Python* as 1.

## 4.2 Tools

Tools used for implementing the neural net model in this work are Pytorch version 1.0.1 and for word embeddings, library used is Gensim (v3.7.2). Other libraries used are pandas (v0.23.4), numpy (v1.14.3) and nltk (v3.3). Tensorflow (v1.13.1) and Tensorboard are used to visualize the network training as well as the accuracy and loss for analysis.

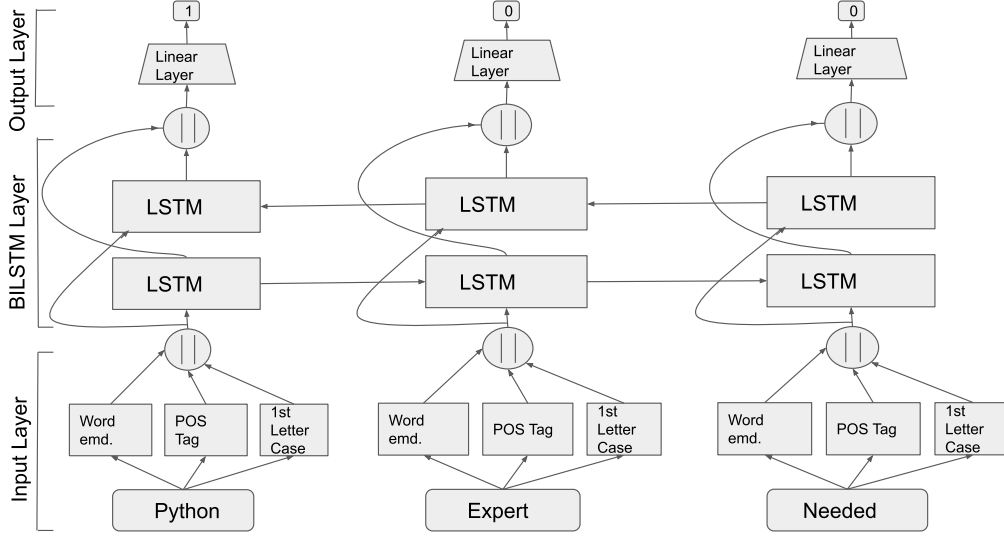


Figure 4.1: Complete BiLSTM Architecture

### 4.3 Data Preparation

As resume data is sensitive, it is not easily available for training the model, and hence for experiments performed in this thesis we use the job descriptions rather than resumes. Job descriptions with all the skills tagged in the text were not easily available on the open data platforms. Online job search portal with having updated skills stack in the job portal itself would have been useful for ensuring tagging of a maximum number of skills present in the job description data. Online job search portal “Stackshare”<sup>1</sup> shows all the tech stacks used by top companies as well as the job positions open for these companies. Thus this was chosen to collect the data for our work. For obtaining the job descriptions, the job ad data was web scraped from Stackshare. For training the model, we needed sentences tagged with skills. As the exhaustive list of skills for this specific portal was not available, the skills list was prepared from the technologies mentioned in the tech stacks in this website. Some rule-based methods were used to extract the skills from descriptions as well. This skill list was then used to tag the sentences automatically by keyword matching. Details of this process are mentioned in the following sections.

<sup>1</sup><https://stackshare.io/>

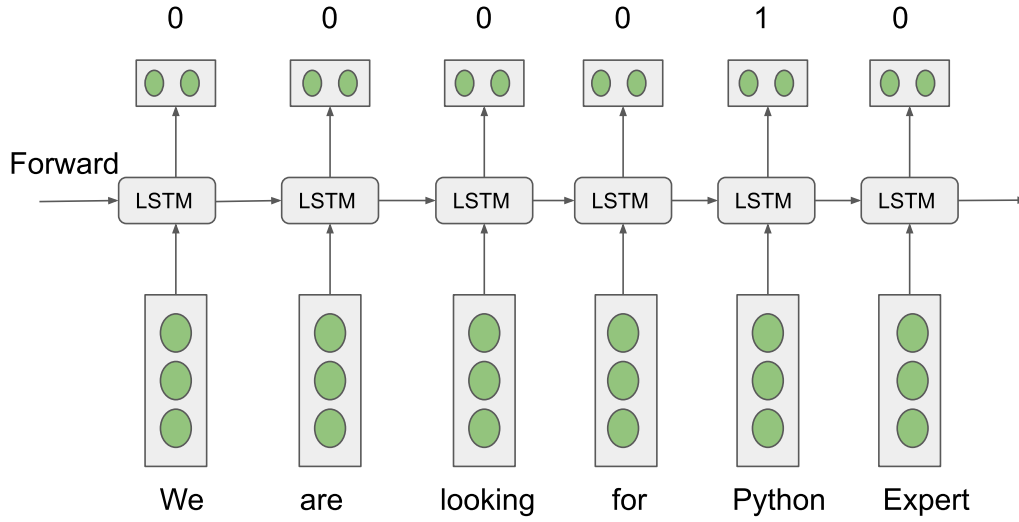


Figure 4.2: LSTM Network for Job Skill Detection

### 4.3.1 Data Collection

Data needed for training consists of the job descriptions and the list of skills in order for labeling in the training phase. From the job descriptions, the labeling phase requires labeling the skills as 1 and the rest of the words as 0. Thus, knowing all the skills is a necessity for proper labeling. Having the exhaustive list of all the skills present in the job description is not guaranteed with the list present from the website nor from the keywords tagged for individual job description in the job ad and thus labeling is not 100% accurate as there might be missing skills from the job description.

Two data sources required were as follows:

1. Job Description
2. Job Skills list

#### Job Skill Collection:

To make this tagging better, the data gathered was focused on one job stack area which is full stack development. The skills list is scraped from the list of stack categories of skills present on the website. Categories found on search (<https://stackshare.io/search?q=full%20stack%20development#>) are as shown in Figure 4.4.

For each category, the list of skills is extracted as to get the complete list of skills for full stack development. For example, clicking on “Languages & Frameworks” the page gives details about the skills under that category for full stack development as shown in Figure 4.5. The figure shows the top



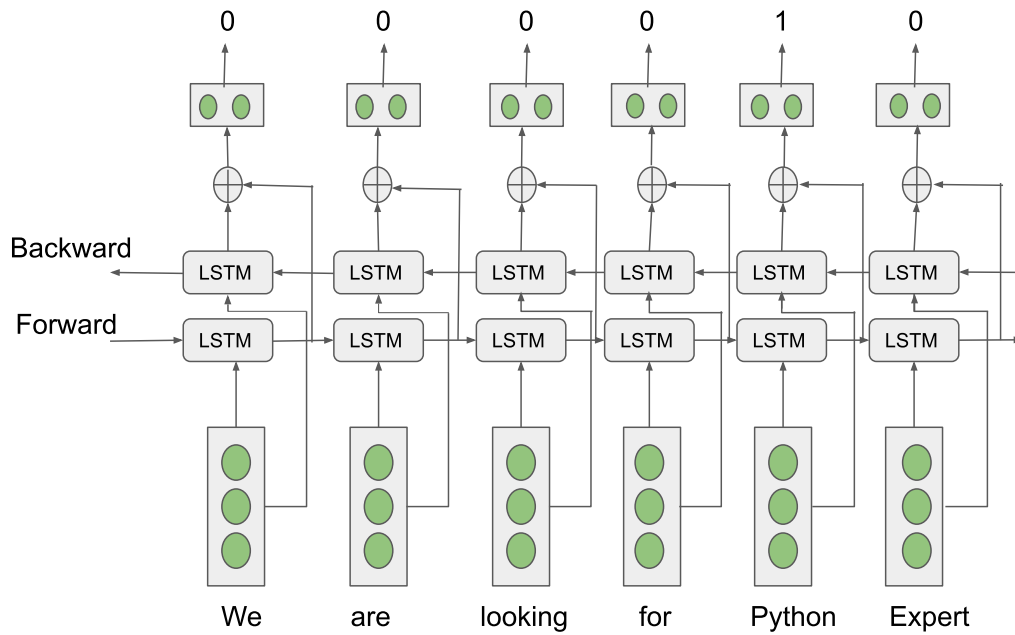


Figure 4.3: BiLSTM Network for Job Skill Detection

skills shown on the page. The names like “Javascript”, “Node.js”, “Python” from the page are extracted from the “span” tags with “itemprop” property as “keywords” gives the name of the skill from the list. This is done for all the categories as shown in Figure 4.4.

#### Job Description Collection:

For extracting the job descriptions, a specific keyword search for skills from the full stack development is done so that the job results would limit the scope to that area only. Top skills were chosen from various categories like databases, backend languages and front-end frameworks to have diverse coverage of descriptions for training. Twenty two skills chosen were “angularjs”, “c”, “c++”, “c-sharp”, “django”, “dot-net”, “go”, “html5”, “java”, “javascript”, “jquery”, “mongodb”, “mysql”, “nodejs”, “nosql”, “perl”, “php”, “postgresql”, “python”, “react”, “ruby” and “spring”. Job search for keyword “html5” will show various job results as shown in Figure 4.6. As not all the job results were shown on the first page, the data was scraped for clicking the load more button 1000 times and getting the descriptions present in the results retrieved from that.

Job descriptions after clicking on each of these job ads were collected for all of these searches. Example of the job description of the first result obtained from the above search (<https://stackshare.io/match/jobs/hostinger-junior-web-developer-php>) is as shown in Figure 4.7.

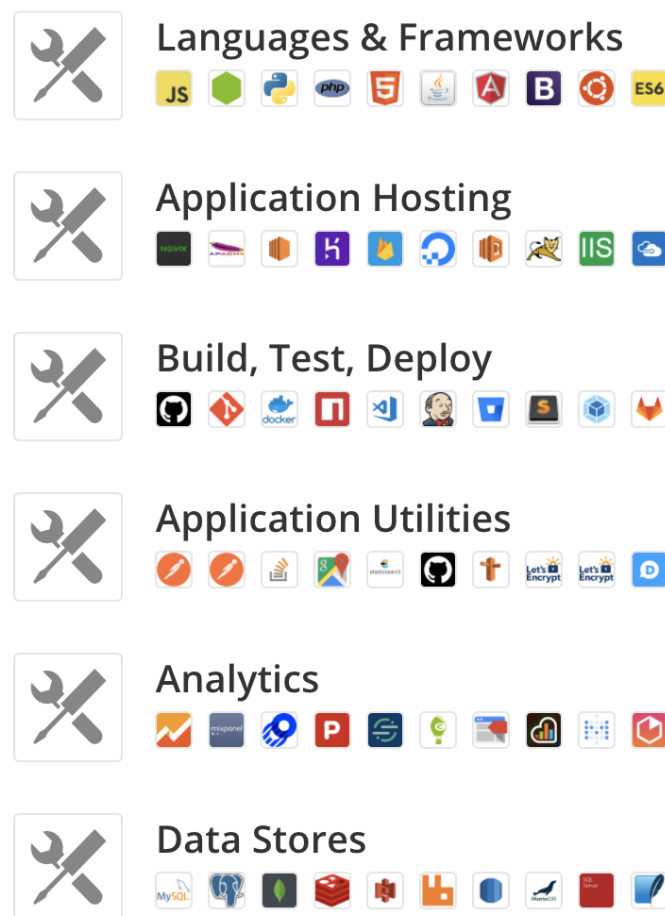


Figure 4.4: Full Stack Development - Stack Categories List

The content of the complete job ad was extracted. The skills listed at the top of the description page were extracted as well to add it to the list of skills in case they were missed from the previously created skills list. The skills were scraped from the “div” HTML tag with a class named “col-md-4 col-md-offset-4” from the page and the description is extracted from the all the paragraph tags from the div with class name “match\_\_job-info\_\_desc”. The total number of job descriptions scraped was 16,934.

### 4.3.2 Data Labeling

For the data labeling task, the skills list obtained from the web scraping was used for tagging the skills as 1 in the job descriptions. The list had many ambiguous words which could also be normal nouns or verbs. For example,

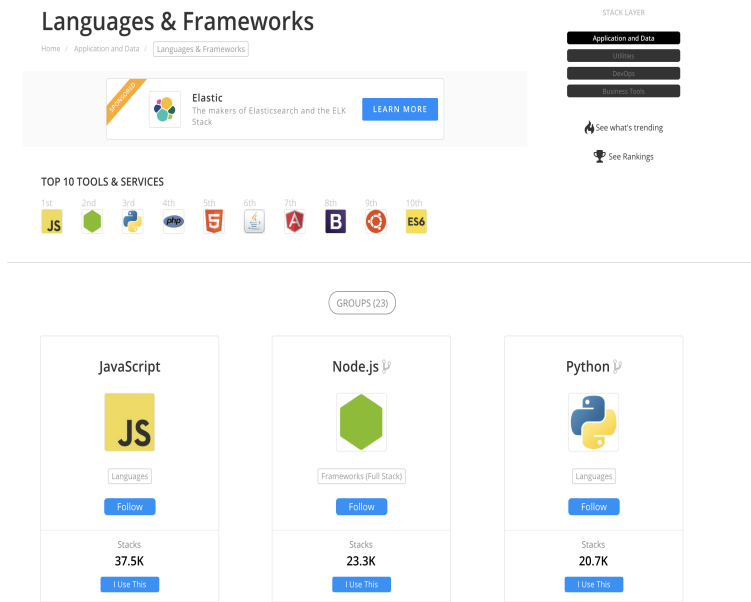


Figure 4.5: Skill List under Languages &amp; Frameworks

there were skills named “go”, “make” and “mean” which can be programming languages or tools. These skill keywords would appear in other sentences with different meaning and thus neural network might learn incorrectly the context if those were tagged as skills in the description. Hence, another list of skills was manually created which should be excluded from the skills list. This was created to give a reduced list of skills. Initially, when the models were run then a lot of predictions happened for the skills which we did not have in the list of skill. Thus those unknown predicted words were extracted from the output of the predictions of the first run on test data. The manual phase of filtering that list was performed to contain only skills. Those new skills found were added to the skills list used for tagging the documents. The purpose of this was that the labeling of skills gets better and there are no missed or wrong labeling done for the skills. Apart from this, a few rule-based methods were used to extract skills appearing in front of the obvious phrases indicating the start of the skill lists from job descriptions. Phrases like “familiarity with”, “working with consists of”, “understanding of”, “you have used”, “such as”, “primarily using”, “experience in”, “experience with”, “knowledge of”, “a plus”, “technologies like”, “programming in”, “stacks”, “basic tech” etc. were searched by keyword matching to extract the ten words after that in the job descriptions. These were manually filtered for finding the skills.

In order to verify that the model is learning the context and able to

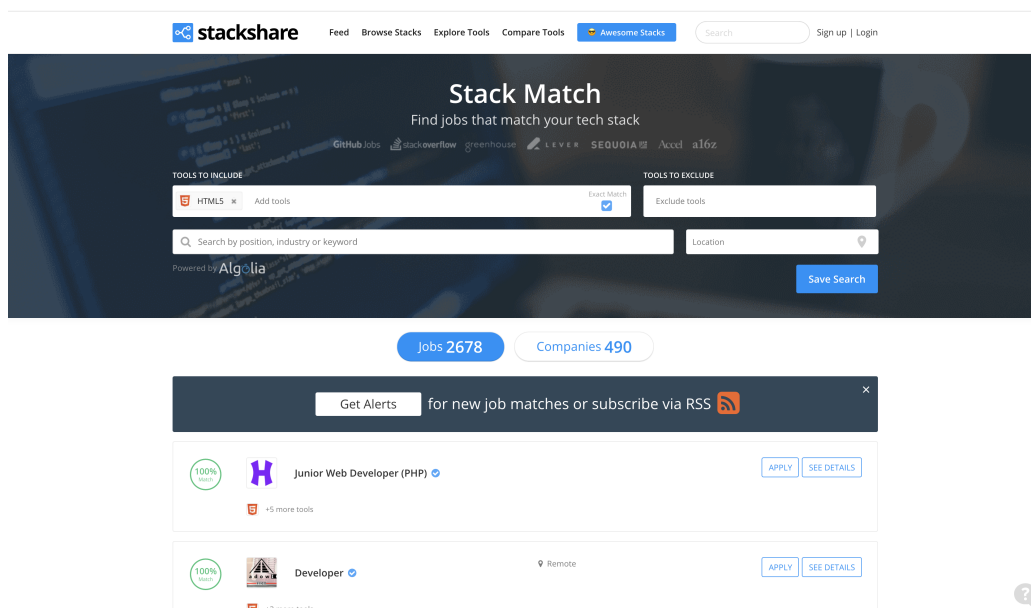



Figure 4.6: Job Search Results for Keyword HTML5

detect skills that it hasn't seen before we set aside some test skills which would appear only in the test documents and not in the train documents at all. Once the skills were extracted, the skills were split randomly into train and test files. Total unique test skills were 1246 and total unique train skills used were 2568.

The data was split into sentences for it to be fed to the neural network. The sentences not containing any skills were excluded. If a test skill appeared in the sentence then it was discarded from the training dataset and were put in the test dataset. Labeling was done where the skill words would appear they were tagged as one and the rest were zero. As not all skills were single words, multi-word skills with length three/two were also considered. In the cases of key phrases spanning multiple words, we tag each of the words by the same skill label (1). This is a weaker form of labeling in comparison to BIO scheme which has a start and end label for key phrases but it is a stringent case for evaluating label accuracy [51]. We have chosen the weaker form of labeling. For example, if the skill was "Linux Mint" then both the words would be labeled as one instead of giving different labels for multi-word skill. During the accuracy or evaluation calculations, the prediction would be counted as correct only if both of them were tagged as 1 by the predictor and the correct skill prediction count would be one instead of two. But if we have "Linux" as skill in the list and "Linux Mint" in the skill list then both matchings would have been done. "Linux" labeled as 1 would be correct

### Junior Web Developer (PHP)



See more tools in Hostinger's stack

APPLY

### Description

Hostinger is looking for a talented Full-Stack Web Developer who loves to create web applications and new features for existing web hosting platforms. If you are a restless web developer who will not stop until you have found the solution to a specific problem, if you care deeply about the quality of your work, and if you have a strong desire to learn and rapidly grasp new technological concepts in a fast-growing, internationally operating company, then keep on reading.

Our next Junior Web Developer will have experience in front-end and back-end technologies and will be up-to-date with the latest web development news and trends. You will be responsible for developing features for existing Hostinger products, and making sure that our 30+ million users across the globe are happy with what we do.

If you are able to listen, understand complex ideas, and make IT happen, then do not hesitate to send us your application!

### Requirements

- Good knowledge of web technologies, plus CSS3, HTML5 and Javascript;
- Experience with the PHP programming language;
- Knowledge of MVC frameworks (preferably Laravel/Zend);
- Working Knowledge of the Git versioning system;
- An understanding of database and SQL;
- Knowledge around the creation and maintenance of web services;
- Ability to think analytically.

Figure 4.7: Job Description Page

prediction and another prediction of “Linux Mint” would also be counted if both these words are predicted with label one. If “Linux” is predicted as 1 but Mint is predicted as 0 then the correct prediction count would be incremented by one and missed skill prediction count be incremented by one as well because it missed “Linux Mint” skill. If both of them are zero then the missed skill prediction count would be incremented by two. This is how the tri-gram/bi-gram skills are handled in the labeling and prediction phase. Labeling done on the basis of the skill list available is shown in Table 4.2.

## 4.4 Input Feature Engineering

As with other problems involving natural language processing, tasks for feature engineering like stemming, tokenization, creating appropriate word representations have been performed to get a suitable input to the network for it to perform efficiently. The input to the network are the sentences from the

Table 4.2: Scenarios for Labeling and Prediction Calculations

Type of skills present in input	Skills known for labeling	Words	Labels for words	Predicted labels for words	Correct skill prediction count	Missed skill prediction count
Single bi-gram skill	“Linux Mint”	Linux Mint	[1, 1]	[1, 1]	1	0
Single bi-gram skill	“Linux Mint”	Linux Mint	[1, 1]	[0, 0] OR [1, 0] OR [0, 1]	0	1
Mixed bi-gram skill	“Linux”, “Linux Mint”	Linux Mint	[1, 1]	[1, 1]	2	0
Mixed bi-gram and single skill	“Linux”, “Linux Mint”	Linux Mint	[1, 1]	[1, 0]	1	1
Mixed bi-gram and single skill	“Linux”, “Linux Mint”	Linux Mint	[1, 1]	[0, 0]	0	2

job descriptions. Each sentence representation can have information about the word, various features like part of speech tag for the word, case of the letters in the word, etc. For the experiments in this thesis, there are a few features selected which would benefit the prediction or which would add information that is indicative of the context for the skill words.

Word embeddings for the words in the sentence are considered as one of the input features. As this is a domain-specific task, already existing Word Embeddings like Glove<sup>2</sup>, Wikipedia word vectors<sup>3</sup>, GoogleNews<sup>4</sup> etc. were not used because they miss the relationship between the words that are more frequent in the job description data. Thus Word2Vec model was trained on all sentences of the job descriptions so that words having semantic similarity would end up having vectors close to each other which would help the prediction ultimately. Gensim module of Word2Vec with Continuous Bag of Words algorithms for training was used to first train the word vectors on the job descriptions corpus and then in the neural network model, the vector obtained from this model was used for training as input. The total unique length of the vocab built with Word2Vec model is of size 48671. Dimensions used for vectors were in the range of the embedding sizes tested for hyperparameter testing which was 200, 300, 400. All these words were mapped to these low dimensional dense vectors using Word2Vec. Another experiment with not using the pre-trained vectors using the Word2Vec model is also tested to measure the performance differences when only pytorch on the fly training is used to learn the word embeddings for the words.

Stemming was not performed on the words for these experiments to avoid the ambiguity between the skill “programming” versus the word “program” for instance. The total number of words in the dictionary was not too huge to

<sup>2</sup><https://nlp.stanford.edu/projects/glove/>

<sup>3</sup><https://fasttext.cc/docs/en/pretrained-vectors.html>

<sup>4</sup><https://code.google.com/archive/p/word2vec/>

accommodate all the words without stemming. While doing the tokenization, sentences were obtained by splitting the job descriptions by newline as well as the sentence ending characters. Also, specific care was taken such that “.” appearing in other instances like website link or email ids etc should not be split and “.” appearing in the versions of software, etc. cases were handled as well.

The Capitalization of the letters was preserved for obtaining the vector for letter case information. While building the vector for indicating if the first letter of the word is uppercase or not, these tokens were used where original cases are preserved. Vocabulary is however built with lowercased tokens obtained from the text. As it was observed in the dataset that many skills are mentioned with the first letter being capitalized. Hence, this would be a good input feature for the model to learn if the word is a skill or not.

Part of speech tagging was also used as one of the features for giving input to the neural network. NLTK package was used to obtain the POS tags for the words using the `pos_tags` function. This was also one of the reasons to not apply stemming to tokens for it to learn the proper tags. Stop words from the text were not removed so that the sequence of the part of speech tagging would help in learning where exactly in the sentence the skills appear. Unique IDs of the part of speech tags are 32 and those ids are used for creating the vector for the sentence as input.

Experiments include different combinations of word embeddings, POS and letter case vectors to see if there is an effect of those features for prediction of the labels. An example input vector for these input features is as shown in Table 4.3. Assuming the IDs for those tags are “NN”: 1, “VBG”: 2, “IN”: 3, “JJ”: 4, “NNS”: 5. The meaning and list of all the POS tags are as per the Penn Tree Bank [33]. The letter case vector is built using the case information of the first letter in the sentence and if it is uppercase then it is tagged as one. The sample example of vector generation is as shown in Table 4.4.

Table 4.3: Part of Speech Vector

Experience	working	in	big	data	systems	such	as	Hadoop	Presto	etc
NN	VBG	IN	JJ	NNS	NNS	JJ	IN	NN	NN	NN
1	2	3	4	5	5	4	3	1	1	1

Table 4.4: Letter Case Vector

[illegible]

## 4.5 Hyperparameter Selection

Optimization and hyperparameter selection play an important role in making the difference between the good and state-of-the-art results with LSTM Networks [50] and it is not straightforward as it requires tuning many parameters like the dropout rate, the pre-trained word embeddings, and many other hyperparameters. Making the correct hyperparameter optimization is often like the “black art that requires expert experiences” [54]. An in-depth analysis in hyperparameter optimization for sequence tagging tasks is published in [50] where the authors have shown which parameters result in high-performance improvements over other parameters.

### 4.5.1 Number of Epochs

The number of epochs initially is set high around 30 to see the performance of the neural network training. A few samples from train data have been set aside to check the performance of the model during training. Model checks the performance during intermediate steps on this set-aside validation dataset. This is used for tuning the hyperparameters of the neural network model [20]. A model may be underfitting if performance on the training set is better than the validation set and performance has plateaued. The over-fitting can be observed when the validation set loss keeps increasing but training loss gets better, thus the training needs to be stopped where the test loss is close to train loss and does not increase with train loss improvement. Thus we check the performance of the model on the validation data to see how well the network is generalizing over that dataset. Plotting the loss for train and validation dataset gives an idea about the model training progress. After running for more epochs, the epoch at which the best performance is achieved is recorded and the final model can be obtained by stopping the training at the appropriate epoch number. After running the model for 30 epochs, the best performance was obtained for the various combinations between 11-15.

### 4.5.2 Word Embedding Dimension

Word embedding dimension between 100-300 is widely used in NLP tasks [50] and it could potentially increase the performance as mentioned in [11]. Thus, these combinations were tested in the experiments.



### 4.5.3 Hidden Layer Dimension

Number of recurrent units has shown not to affect the network performance largely. Having this value too large or too small will not be suitable either [50]. If it is too small it will not be able to store the necessary information and if it is too large then it will overfit the train data and the test performance will decrease. A value of 100 has been shown to give good results in POS, NER, chunking, Entities detection and event recognition which involves sequence labeling [50]. Thus, in this thesis experiments, we have used 100 hidden units.

### 4.5.4 Mini Batch Size

Mini Batch gradient descent is a variation of the gradient descent algorithm that divides the training data into small chunks of batches that are further used to calculate model error and to update model coefficients. Gradient descent is used for optimizing the model and there exist different types like Stochastic, Batch and Mini Batch variations of the Gradient Descent algorithm. Mini-batch updates the model more frequently than the Batch version and thus it allows for more robust convergence and helps to avoid local minima. For the POS tagging task, the optimal size of 1 was found [50]. Thus, the choice of mini-batch size of 1 was selected. The experiments were run without padding the sentences. If padding is done in the future, then tests for mini-batch sizes up to 8-16 can be tested to see the performance differences.

### 4.5.5 Optimizers

Adam [3] and Adam with Nesterov momentum [37] (Nadam<sup>5</sup>) optimizers have performed the best for sequence labeling tasks using LSTM networks, followed by RMSProps [63] [50]. Nadam had the best convergence time in [50]. SGD [60] has shown high sensitivity to learning rate. The adaptive learning rate for Adam has performed well. Thus for the experiments, Adam with adaptive learning rate, Nadam and SGD with a few different learning rates were used to find the best performing optimizer and experiments were performed on these optimizers to observe if it showed similar performance results as shown in this research for sequence labeling task [50].

---

<sup>5</sup><https://github.com/rwightman/pytorch-commands/blob/master/train.py>

### 4.5.6 Dropout

Dropout helps the neural network to generalize better [37]. Dropout multiplies neural net activations by random zero-one masks while training. The dropout probability  $p$  determines what portion of those mask values are one [5]. The dropout value of 0.5 works well for a wide range of tasks [58] but the speech community has experienced that careful selection of the dropout is required for speech related tasks [5]. Thus, even for our task involving language understanding we experiment and find out which one works the best for this task. In dropout, there is an option to not use dropout, use naive dropout or use variational dropout. Variational dropout performed the best on all tasks of sequence labeling used in [50] superior to no-dropout or naive dropout. Due to time constraints, only naive dropout strategy with values [0.05, 0.1, 0.25, 0.5] was tested to find the most effective values.

## Chapter 5

# Evaluation

In this chapter, we evaluate the different techniques mentioned in previous chapters for extracting skills from the job descriptions. A comparison of different techniques and analysis of the results is done at the end, to determine which method works best for this task.

### 5.1 Quality Metrics

All approaches experimented are evaluated primarily on the precision, recall, F1 Score and accuracy attained for test and train skills. Accuracy of test and train skills is done separately to understand the out of vocabulary skill detection capacity of the system. Out of vocabulary is the list of words that the model has not seen in the training dataset. This is an essential factor for evaluation since this proves that the model is actually learning the context and not just the word itself. This is the metric that would determine the suitability of the system in automatically detecting the unseen skills. Simply counting the words labeled as skills is not sufficient since there are multi-word skills present as well. Hence, the correctly predicted skill is as per the explanation is given in Section 4.3.2 of Data Labeling. Once the correct number of skills are predicted and missed skills predicted count is obtained, true positive, false positive and false negative are calculated and then calculations for precision, accuracy, etc. are done. Train and test skill lists are split in the beginning as the evaluation would involve considering two cases of evaluation: 1. train skill performance and 2. test skill performance. The counts of correct prediction of skills are counted separately for each of these cases to get the accuracy for the corresponding category. If the word is predicted as 1 (skill) but it does not exist in any of the above lists then it is counted in unknown skill prediction counter. Precision, Recall and F1 scores

need calculation of true positive, false positive and false negative which is done as given in following equations equations 5.1, 5.2 and 5.3:

$$\text{true\_positive\_count} = \text{correct\_test\_skills\_predicted} + \text{correct\_train\_skills\_predicted} \quad (5.1)$$

$$\text{false\_positive\_count} = \text{unknown\_skill\_predicted} \quad (5.2)$$

$$\text{false\_negative\_count} = \text{missed\_test\_skills} + \text{missed\_train\_skills} \quad (5.3)$$

Precision, recall and F1 score are calculated as per the formulae given in 5.4, 5.5 and 5.6:

$$\text{precision} = \text{true\_positive\_count} / (\text{true\_positive\_count} + \text{false\_positive\_count}) \quad (5.4)$$

$$\text{recall} = \text{true\_positive\_count} / (\text{true\_positive\_count} + \text{false\_negative\_count}) \quad (5.5)$$

$$\text{f1\_score} = 2 * (\text{precision} * \text{recall}) / (\text{precision} + \text{recall}) \quad (5.6)$$

Another metric measured is the average efficiency of the skill detection which is the average percentage of the skills detected out of skills actually present. Average efficiency is counted by calculating the percentage of skill detection rate per line and taking the average based on the number of rows in which it is present. This is done separately for train and test skill to use that as a metric for the performance of the system. It indicates given the number of skills present in the sentence how many will be detected on average by the system built. Efficiency per line is calculated using formulae given in equations 5.7 and 5.8 given as follows:

$$\text{test\_skill\_efficiency\_per\_line} = (\text{correct\_test\_skills\_predicted} * 100) / \text{total\_test\_skills\_present} \quad (5.7)$$

$$\text{train\_skill\_efficiency\_per\_line} = (\text{correct\_train\_skills\_predicted} * 100) / \text{total\_train\_skills\_present} \quad (5.8)$$

Once this is calculated per line and are summed up. Then the average efficiency is calculated for test using Equation 5.9 and for train using Equation 5.10 which are given as follows:

$$\text{total\_test\_efficiency} = \frac{\text{sum\_of\_test\_skill\_efficiency\_per\_line}}{\text{number\_rows\_having\_test\_skills}} \quad (5.9)$$

$$\text{total\_train\_efficiency} = \frac{\text{sum\_of\_train\_skill\_efficiency\_per\_line}}{\text{number\_rows\_having\_train\_skills}} \quad (5.10)$$

This is calculated separately for train and test skills. For example, the total efficiency is 70% for train skills which indicates that for every train skill present in the line has a 70% chance of getting detected as a skill by the system.

## 5.2 Results

Methods used for evaluation were quality metrics mentioned above for comparison. To monitor the model training, test vs. train loss and accuracies were plotted to find at which epoch both the train and test loss stabilizes. To visualize the loss, train and test loss at an equal interval of 10,000 steps were noted and plotted as shown in Figure 5.1. Model is trained on 60,756 documents for 15 epochs and after every 10,000 steps model is evaluated on 5000 validation documents. Train loss and validation loss is recorded at these points. The train and test loss plotted for the best hyperparameter combination obtained for BILSTM experiment is shown in Figure 5.1. The plot depicts that the model was learning and improving the predictions as the loss showed a steady decreasing trend. Along with train loss, validation loss is also decreasing which shows the model is not overfitting the train data. After step 45 the training loss has plateaued but test loss is improving slightly after that, hence training the model for few more steps has improved the model. This is observed to find out the ideal number of epochs initially. 15 epochs were found to be sufficient for this experiment to get reasonable loss values. Similar charts plotted for various combinations helped to verify if the model is not underfitting or overfitting.

### 5.2.1 Hyperparameter Selection

Hyperparameter selection for the neural network was obtained by running multiple experiments with different settings of the learning rate, optimizers,

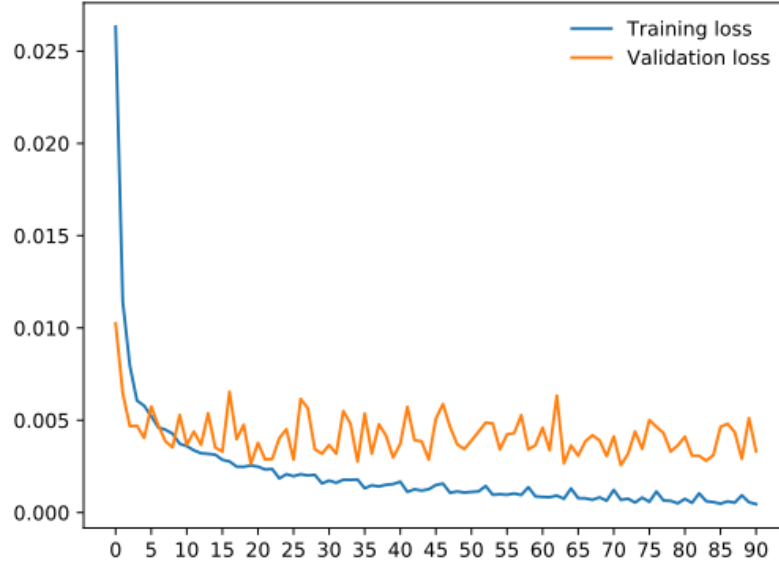


Figure 5.1: Train loss Vs Validation loss

dropout and learning rate updating techniques. Once the optimal parameters were found, LSTM and BiLSTM performances were compared by fixing these hyperparameters with the best parameters discovered from these experiments. For testing the hyperparameters, BiLSTM classifier with all the features (word embeddings, POS and case tag) were used to first find the hyperparameters and then those best parameters were used for comparing other combinations of inputs and classifiers that are comparing the LSTM and BiLSTM classifiers with different combinations of input features. First, the type of optimizer and type of learning rate updating techniques were selected for experiments by running it for optimizers Adam, Nadam, RMSProps and SGD. As previously discussed in Chapter 4 these have proven to work well for a sequence labeling task.

Adam optimizer has been shown to work well with the Adaptive learning rate, thus we experimented with two learning rates for the Adaptive setting for Adam optimizer first. We also ran it with Non-Adaptive learning rate updating technique. The Adaptive learning rate is implemented with `stepLR`<sup>1</sup> function in pytorch which decays the learning rate at every step size(=5) by

<sup>1</sup>[https://pytorch.org/docs/0.3.0/\\_modules/torch/optim/lr\\_scheduler.html#StepLR](https://pytorch.org/docs/0.3.0/_modules/torch/optim/lr_scheduler.html#StepLR)

$\gamma (=0.1)$  which are set manually in the parameters of the function. The experiments were run for 15 epochs and the step size used was 5. When the starting learning rate is 0.01 then the following learning rate values at different step size intervals will be 0.001, 0.0001 and so on. From the Table 5.1, 0.01 worked better compared to 0.05 for Adam. Adaptive strategies with different starting learning rates were tested for Adam as that was shown to work well for that optimizer [50]. Along with optimizers, different embedding dimensions were tested in parallel as that affects the performance too [50]. Embedding sizes of 50, 100 and 200 were tested for each optimizer and learning rate updating technique.

The learning rate of 0.01 was kept constant while comparing different optimizers. Non Adaptive learning rate in the majority of the cases worked better for Adam, Nadam, and SGD. The best results obtained in these combinations for Adaptive and Non-Adaptive are plotted as shown in Figure 5.3. Results show that the best results for different optimizers had obtained from the Non-Adaptive learning rate technique except for Nadam were both were equally performing well. Even though the best results were the same for Adam, the embedding size required to get the same result in the Adaptive strategy was 50 as opposed to 200 in Non-Adaptive strategy. Results obtained in [50] had suggested that Adam worked well with adaptive but in our results, Non-Adaptive worked better than the Adaptive when comparing the best test skill accuracy metric. RMSProps with 100 embed size and Non-Adaptive learning rate worked better than the best score obtained with Nadam and Adam which is not similar to what was found in [50] where Nadam gave the best results on sequence labeling. For this specific dataset, SGD with Non-Adaptive 0.01 learning gave the best results with 56% Test skill accuracy and F1 score of 82. This is obtained with a 50 embedding dimension compared to other dimensions. Test skill accuracy metric is used for comparison for these experiments since the ability of the system to detect the words not seen in the training dataset is an important indicator for systems requiring to deal with unseen words. This will be indicative of the system's performance on identifying skills automatically and thus considered as the main differentiating factor while choosing the best hyperparameters.

When the best optimizer and learning rate updating technique was obtained, those parameters were fixed and then the comparison of pre-trained word embeddings with non-pre-trained word embeddings was done. The results clearly showed significant differences between the pre-trained and non-pre-trained word embeddings. The highest test skill accuracy obtained without pre-trained word embeddings was 14% compared to 56% when pre-trained word embeddings were used.

When the best optimizer and learning rate updating strategy was ob-

Table 5.1: Hyperparameter Optimization Results

Pre-trained Word Embeddings	Word Embedding Dimension	Optimizer	Type of LR	LR	(F1, precision, recall)	Test Skill Accuracy(%)	Train Skill Accuracy(%)
Yes	50	Adam	Adaptive	0.05	(52, 90, 37)	20	48
Yes	100	Adam	Adaptive	0.05	(62, 89, 47)	24	61
Yes	200	Adam	Adaptive	0.05	(61, 85, 48)	27	60
Yes	50	Adam	Adaptive	0.01	(75, 91, 65)	39	80
Yes	100	Adam	Adaptive	0.01	(74, 92, 61)	34	78
Yes	200	Adam	Adaptive	0.01	(74, 91, 63)	35	80
Yes	50	Adam	Non-Adaptive	0.01	(75, 89, 65)	41	80
Yes	100	Adam	Non-Adaptive	0.01	(72, 90, 60)	33	77
Yes	200	Adam	Non-Adaptive	0.01	(76, 89, 66)	40	82
Yes	50	Nadam	Adaptive	0.01	(74, 91, 62)	36	78
Yes	100	Nadam	Adaptive	0.01	(69, 92, 56)	28	72
Yes	200	Nadam	Adaptive	0.01	(72, 92, 59)	31	75
Yes	50	Nadam	Non-Adaptive	0.01	(69, 91, 56)	30	71
Yes	100	Nadam	Non-Adaptive	0.01	(71, 90, 58)	30	75
Yes	200	Nadam	Non-Adaptive	0.01	(74, 90, 63)	36	79
Yes	50	RMSProps	Adaptive	0.01	(78, 89, 70)	46	84
Yes	100	RMSProps	Adaptive	0.01	(79, 90, 70)	45	86
Yes	200	RMSProps	Adaptive	0.01	(80, 90, 72)	46	88
Yes	50	RMSProps	Non-Adaptive	0.01	(78, 88, 69)	44	85
Yes	100	RMSProps	Non-Adaptive	0.01	(80, 87, 73)	50	88
Yes	200	RMSProps	Non-Adaptive	0.01	(60, 92, 45)	19	60
Yes	50	SGD	Adaptive	0.01	(79, 89, 71)	44	88
Yes	100	SGD	Adaptive	0.01	(77, 90, 68)	39	85
Yes	200	SGD	Adaptive	0.01	(79, 90, 70)	43	86
<b>Yes</b>	<b>50</b>	<b>SGD</b>	<b>Non-Adaptive</b>	<b>0.01</b>	<b>(82, 85, 79)</b>	<b>56</b>	<b>94</b>
Yes	100	SGD	Non-Adaptive	0.01	(81, 86, 77)	52	93
Yes	200	SGD	Non-Adaptive	0.01	(82, 87, 78)	55	92
No	50	SGD	Non-Adaptive	0.01	(63, 89, 49)	5	74
No	100	SGD	Non-Adaptive	0.01	(70, 87, 59)	13	86
No	200	SGD	Non-Adaptive	0.01	(65, 60, 72)	14	87

tained, the next hyperparameter selection experiments for finding optimal learning rates were done. Learning rates  $\{0.001, 0.004, 0.01, 0.02, 0.025, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09\}$  were tested with best optimizer SGD and Non Adaptive learning rate updating strategy with 50 embedding dimension is as shown in Table 5.2. Values far from 0.01 in both directions did not give comparable results for test skill accuracy. As shown in Figure 5.2 peak is found at 0.01 learning rate when test skill accuracy for all learning rates is plotted. Accuracy rates showed an increasing trend after 0.04 learning rate but they were not as high as 0.01 learning rate. The values of test skill accuracy rates are not constant for every run and thus to avoid biased interpretations based on a single run, experiments were run twice and average values of two runs were plotted. These experiments confirmed that the learning rate chosen was optimal and no other optimal learning rate was found from the experimented learning rates. Another hyperparameter, dropout was tested with different probability values on the LSTM layer. Dropout is a good way to generalize the model [5] and thus the model was tested with



different dropout values for observing the performance variations based on that is as shown in Table 5.3. Dropout values experimented were  $\{0.01, 0.05, 0.1, 0.25, 0.5\}$ . As can be seen from the table, dropouts did not improve the results compared to when the dropout was not used. Test skill accuracy with a dropout of 0.25 was closer to the best result of 0 dropouts. This shows that adding this dropout for LSTM layers did not reduce the performance much and thus can be used if a more generalized model is required. For further experiments, we keep the dropout of 0 as that gave the best results.

Table 5.2: Learning Rate Performance comparison for Best Optimizer

Learning Rate	Average Test Skill Accuracy	Average Train Skill Accuracy
0.001	44	84
0.004	44	86
<b>0.01</b>	<b>52</b>	<b>91</b>
0.02	47	88
0.025	44	88
0.03	44	87
0.04	43	89
0.05	46	88
0.06	45	87
0.07	48	90
0.08	48	89
0.09	52	89

### 5.2.2 Comparison of Input Features and Models

Results in Table 5.4 indicate that BILSTM with all the three features performed better than the rest of the combinations of values. Adding the features has not shown significant changes in the performance for the LSTM

Table 5.3: Dropout Performance comparison for Best Optimizer

Dropout	(F1, precision, recall)	Test Skill Accuracy	Train Skill Accuracy
0.01	(81, 88, 74)	49	90
0.05	(81, 88, 74)	49	90
0.1	(80, 87, 75)	49	91
0.25	(81, 88, 76)	52	90
0.5	(80, 89, 72)	48	87
<b>0</b>	<b>(82, 86, 79)</b>	<b>56</b>	<b>94</b>

model. There is a marginal improvement after using the POS tags and case tags for the BILSTM model. Adding more features on top of the pre-trained word embeddings has certainly improved the results for BILSTM. BILSTM has shown to give better results than LSTM. F1 score is more or less the same for both LSTM and BILSTM. As we are interested in finding out how many test skills were detected, BILSTM has shown to detect 56% of the test skills compared to 51% in LSTM. This metric is used for comparing the performance and based on that BILSTM performed the best for job skill detection compared to LSTM.

Table 5.4: Performance Metrics for all experiments

Features	Model	F1 Score	Precision	Recall	Test Skill Accuracy	Train Skill Accuracy
Word Embeddings	LSTM	81	87	75	51	91
Word Embeddings + POS	LSTM	81	88	75	51	90
Word Embeddings + POS + Case Info	LSTM	80	88	74	48	90
Word Embeddings	BILSTM	81	87	76	51	93
Word Embeddings + POS	BILSTM	80	88	72	47	88
<b>Word Embeddings + POS + Case Info</b>	<b>BILSTM</b>	<b>82</b>	<b>85</b>	<b>79</b>	<b>56</b>	<b>94</b>

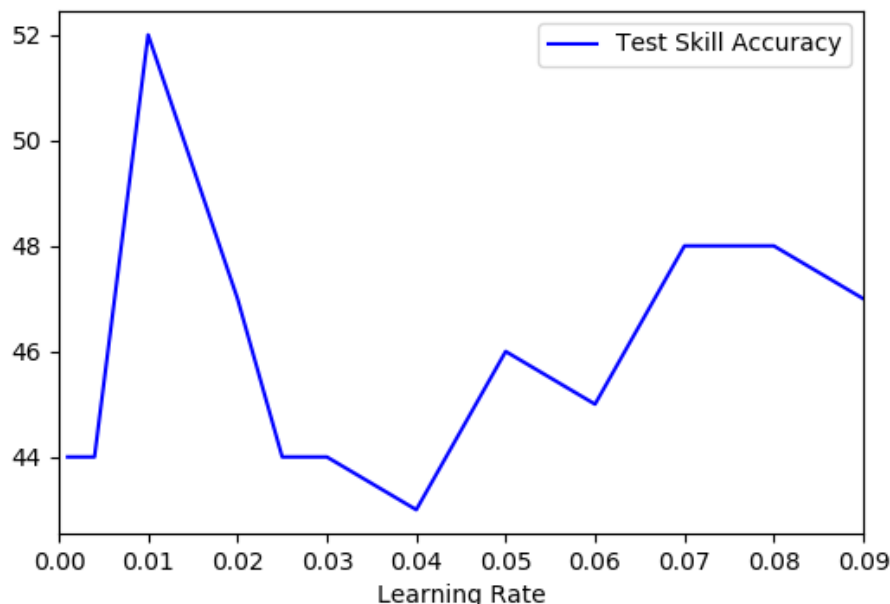


Figure 5.2: Learning Rate Vs. Test Skill Accuracy

For representing the output of the predictions, an HTML file was created to visualize the outcome of predictions in a better way. The sample output of the best model prediction output file is as shown in Figure 5.4. Prediction for the first sentence in the figure shows that [“iam”, “route”, “dynamodb”, “redshift”, “unix”] are some unknown words(not present in our skill list) that are predicted as 1 which we can easily see that those are actually skills. Words [“aws”, “elasticache”, “emr”, “sqs”, “elb”, “puppet”] are the test skills which have been predicted correctly by the model. The word “designing” is a test skill that is missed but from the sentence, it is clear that it is not a skill in this context as the main skills are “unix” and “linux”. Word “vms” is in the train skill list which it failed to detect as skill. Words [“s3”, “cloudfront”, “rds”, “redis”, “sns”, “route53”, “bare”] are train skills which have been predicted by the model correctly. Some more examples are shown in Figure 5.5. This figure shows unknown predictions like [“redshift”, “dynamodb”, “pyramid”, “oracle”] are indeed skills and many of the test skills which are in sequence of skills are predicted as skills as well. Word “oracle” was omitted from the skills list as it is the name of the company as well. The model has predicted it as a skill in where it is actually a skill and not as a company name mentioned in the sentence.

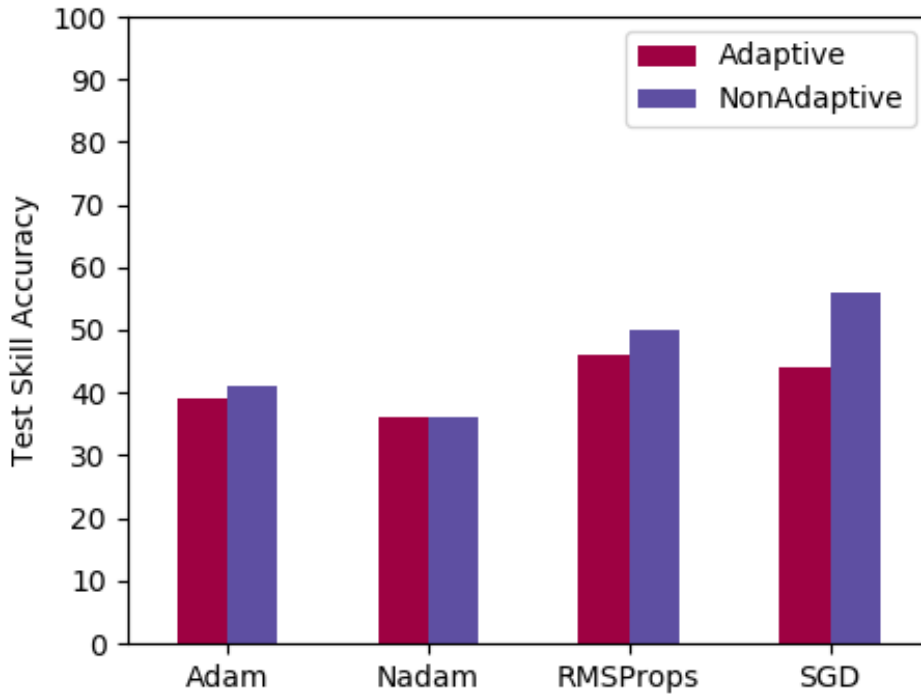


Figure 5.3: Optimizer wise comparison of Test Skill Accuracy by Learning Rate Updating Strategy

Final results for the model resulted in an accuracy of 94% for train skills and 56% for test skills which is an indicator of the BILSTM model being capable of capturing the context to learn the skills present in the job description sentences. Total train skills present in the test dataset was 1,34,070 out of which 1,25,775 skills were successfully predicted. Out of 84,185 test skills present 47,035 skills were predicted correctly. The analysis of unknown skills predicted showed that out of 31,492 skills 2000 were unique. After manual inspection, it was found out that around 1800 of those were actual skills and 200 were false positives. This adds to the accuracy of predicting the unknown skill in the dataset. Overall the performance of the system has shown to detect around 60% of the skills that were not shown to the system in the training.

End of Row\*10362\*

Row: \*10363\* extensive experience in **designing** \*\*\***unix**\*\*\* **linux** system architecture

End of Row\*10363\*

Row: \*10364\* solid **aws** experience including following services \*\*\***iam**\*\*\* **vpc** \*\*\***route**\*\*\* **s3** **cloudfront** **rds** \*\*\***dynamodb**\*\*\*  
\*\*\***redshift**\*\*\* **redis** **elasticache** **emr** **sns** **sqs**

End of Row\*10364\*

Row: \*3320\* our infrastructure includes **aws** **ec2** thousands of **vms** **route53** **elb** **s3** **bare** metal lots of exotic hardware in exotic places  
with exotic vendors all playing nice with **puppet**

End of Row\*3320\*

Figure 5.4: Prediction Sample Output with All Cases of Predictions. Bold RED is for missed train skill, BOLD GREEN is for correct train skill prediction, NORMAL GREEN is for test skill prediction, NORMAL RED is for missed trained skill and BOLD BLACK surrounded by  is unknown skill prediction

Row: \*24\* \*\*\***redshift**\*\*\* \*\*\***dynamodb**\*\*\* **rds** data pipeline **api** gateway and **elasticache**

End of Row\*24\*

Row: \*25\* experience with other cloud based big data tooling and a demonstrable understanding of the pros and cons in applying these  
preferably google cloud platform **bigquery** **dataflow** app engine **cloudera** **spark**

End of Row\*25\*

Row: \*26\* education **bs** in computer science related degree or equivalent experience

End of Row\*26\*

Row: \*27\* other **python** web frameworks as web2py **flask** \*\*\***pyramid**\*\*\* **tastypie** etc

End of Row\*27\*

Row: \*28\* other relational **databases** **mysql** **mariadb** \*\*\***oracle**\*\*\* etc

End of Row\*28\*

Figure 5.5: Prediction Output with Unknown and Known Skill Prediction

## Chapter 6

# Discussion

### 6.1 Results Interpretations

Results obtained for skill detection using BILSTM (Test Skill Accuracy=56%,  $F1 = 82$ , Train Skill Accuracy= 94%) were comparatively better than LSTM (Test Skill Accuracy= 48%,  $F1 = 80$ , Train Skill Accuracy= 90%). This indicates that using the context before and after the word were indicative of the skill presence and has improved the results. The possible reason behind this could be that there are many words that occur right after the skills which make the prediction of the word skill better than LSTM. For example, in job description sentence “Experience with Virtualization and Containerization is a plus” where words [“is”, “a”, “plus”] after the skills *Virtualization* and *Containerization* are indicative of the previous words being skills as much as the words before the skills [“Experience”, “with”]. There are different ways of expressing the same needs and possibilities of contexts appearing after the skills which are quite likely since different recruiters would use different styles of expressing job requirements. Hence, BILSTM certainly utilizes the context on both sides and does not miss out on important indicator contexts for skill words. Results obtained are aligned with this hypothesis that BILSTM should not give worse results than LSTM if not better since BILSTM utilizes more information for prediction than LSTM.

Test skills prediction was improved after training the Word2Vec model on all the documents, this is another requirement that the word embedding learning should be learned on the new data to get better skills predictions. This suggests that a sufficient amount of examples for new skills should be present before trying to predict the new documents containing new skills. The difference of test skill accuracy between the pre-trained and non-pre-trained word embeddings was significant. The highest accuracy for

test skills obtained was 14% with non-pre-trained compared to 56% in the case of pre-trained. This indicates that word embeddings which represent the semantics of the words present in the corpus are an essential feature for the BILSTM/LSTM model to work well on the unknown words.

Another observation is also that only having the context of the words is not enough for prediction. Since the word itself also has some importance as the common stopwords or most frequently occurring words like “other”, “etc”, and “so on” which could appear in the sequence of skill words where skills are listed one after the other. For example, “We need a person having skills in Python, Java, SQL and MongoDB” has skill words one after the other in the sentence so the model could learn if the word is surrounded by skills then the word under consideration is a skill. However, this is not the case every time as there exist sentences for example “We need a person having skills in Python, Java, MongoDB, etc.”. If the model is learning that “Python”, “Java”, “MongoDB” etc. are always followed by each other then the word present in that sequence will be predicted as skill given the neighboring words are skills. Also, the word embedding vector of non-skill word appearing closer to skills could also be similar to skills’ word vector since it often appears in that context. Hence, having additional features other than just word embedding will help distinguish skills better for the model. The reason behind correctly tagging such cases from our best model could be that these additional features might have helped to understand the skill features appropriately. There are cases where the model has properly tagged the non-skill words as 0 when occurred close to skill sequence. For instance “daily work with ubuntu debian or other linux distributions” in this *other* is not tagged as a skill by model and [“ubuntu”, “debian”, “linux”] are correctly tagged as skills. This is a very frequent sequence that is present in job descriptions where requirements are listed in sequence.

There is a possibility of misleading predictions when only context and word embeddings are considered as there are conflicting situations when non-skill has the same context as skill. For example, “fluency in french english and other languages is a plus” and “fluency in developing both object-oriented and functional code on server and client”. In the prior example “French” is a skill and in later example word “developing” is not a skill. BILSTM/LSTM model given only the word embeddings for words in the input sequence will not be able to make the distinction. We can observe that for the best model parameters without the POS tagging used as input it fails to detect French as a skill but with POS added as feature it detects it as a skill since the POS tags for prior example *french* is “JJ” tag and in later case *developing* has “VBG” tag. This feature difference would help the model learn when this “JJ” is followed after “Fluency in” word sequence then it is a skill otherwise

it is not. This shows the importance of adding extra features for resolving such conflicts.

Automatic labeling of the skills had missed a few actual skills present in the descriptions. Since some of the skills were excluded from the skills list those were not tagged as skills while feeding it to network for training. The model has missed learning those contexts which could have been good predictors for skill detection. Many of those words and some unknown skills were predicted by the model on test dataset which has shown the capability of the model to predict unknown skills based on the context. One of the reasons could also be that context for those unknown words had been learned due to the presence of the samples having actual skills with the same context. For example, the context of “familiar with” was often present in the job descriptions and thus when the skipped skill of “node” from the labeling skill list was present in the test dataset with this context then it was predicted as a skill by the model. Word “node” when used with “node js” is a skill but individually it could not be a skill thus was removed from the list. The model detects it as skill when it appeared in the sentence “familiar with node js, hapijs, mongo db”. This is one of the reasons why unknown words predicted as skills when inspected manually were found to be actual skills.

## 6.2 Comparison with Related Work

Related research in the area of sequence labeling for entity recognition in resumes had used manual labeling which had given the F1 score of 76 [28]. In our experiments, the labeling was automated due to a large number of documents and thus was not 100% accurate. Even with those limitations, the results gave an F1 score of 82. Skills not present in the training dataset were also tested and accuracy obtained for that was 56%.

Research done on building the skills tagging for the CareerBuilder [74] where they had used 100M documents for training had obtained an accuracy of 82% and recall of 72%. In this thesis, we obtained a total 79% accuracy with 65K input train data. 85% precision and 79% recall for skills is obtained with this small dataset. Given the constraints of limited skill tagging for training and the amount of data, the accuracy and recall is comparably better. Limited skill tagging due to automation and unavailability of all the skills was the main hurdle in obtaining completely reliable labeled data for our experiments. Even if all the skills were present, fully automated labeling is not possible due to the conflicts between the skill words and their meaning in other contexts. For instance, there are skill names like “MEAN”, “GO”, “ARM” etc. which have other meanings too and could occur in sentences



where they are not skills. There are instances like company names which could be skill too. This is more common in software-related jobs since companies make software technologies. Tagging all the occurrences of these words would have resulted in the wrong training of the model since it would learn the contexts where it is not a skill. To avoid this scenario we had excluded the skills from the labeling. On the other hand, it is also misleading training data for the NN because there are cases where the word is a skill, but the training label is 0.

### 6.3 Limitations of Experiments

Good labeling is crucial to model training to be able to learn the skill contexts correctly. Automatic labeling was done in the experiments which do not cover all the skills. In the cases where the labeling was done properly by the automated labeling, those cases were predicted mostly. There are also cases where the words are actual skills but were excluded due to their ambiguity. Some of the ambiguous words which are in some cases skills and not in others were excluded from the skills lists. These ambiguous words are like company names, words which have multiple meanings, etc. Some of those were missed by the system. This is because manual labeling was not done. Some of the examples with corresponding reasons are as shown in Table 6.1.

Table 6.1: Missed Skills in Training Data

Sentence	Examples of Missed Skill Words	Reason
developing visualization dashboards interfacing with tools like <b>tableau</b>	tableau	company name
experience working with one or more of the following apis <b>facebook salesforce</b> saml	facebook, salesforce	company name
our stack covers <b>go</b> scala java kafka grpc protobufs and aws	go	multiple meanings
you are a wizard on the <b>mean</b> stack	mean	multiple meanings

As all skills were also not available, there were also cases where the skills were not tagged in the training dataset. This limitation can be handled by adding more features other than word embedding which is specific to skills in order to capture more of these mislabeled skills in the training data. Adding just the POS tags is not enough since, with a limited number of POS tags, the skill words cannot be distinguished from other words. Observation of the job descriptions showed that most of the skills have the first letter capitalized thus combining both these features have given marginally better performance over just the word embeddings. Lack of proper labeling is certainly one of the limitations of this system.

Neural net models take a lot of time for training, thus due to the limited time, the number of epochs ran was around 15. These are practical limita-

tions even in real time applications where it is not possible to get the labeled data. Apart from that, there exist many combinations of hyperparameters. This makes it difficult to state that the hyperparameters obtained from selected choices of combinations for experiments would yield the most optimal combination.

Pretrained word embeddings were trained on a complete corpus which had test data as well which might have helped in predicting the unseen words. It was necessary to do the training of Word2Vec on all the documents including the test documents because otherwise the word embeddings would not have been learned for the words that exist only in test data. In real-world applications, sufficient amount of new skill instances must be present or the assumption is that those would have a similar context as given in some of the training datasets. If these conditions fail then the system might not be able to predict the skills. False positives which are not actual skills are also one of the concerns for this automatic detection as they cannot be avoided completely and the current system predicts some words which are not skills as skill.

## 6.4 Future Work

Similar research in the area of finding attributes from the description data [51] had obtained better results with the CRF approach on top of using BiLSTM. Their experiments resulted in 91% accuracy for previously unseen attributes. This is more compared to the results that are obtained in this thesis. Possible reasons could be the use of CRFs based approach as well as the proper labeling done before training and using a large number of documents for training which was not feasible for the study done in this thesis. Character embeddings along with the word embeddings were used in their research. There are a few studies done where character level features extracted using CNN based approach were used as input to sequence labeling model [36]. Apart from LSTM recurrent unit, GRU units have shown to work comparable to LSTM for sequence modeling task [10]. This approach can be compared with LSTM and BiLSTM based sequence labeling. All the above-mentioned approaches could be tested in the future to improve the results obtained in this thesis.

In one of the studies related to sequence tagging, connecting the extra spelling and context features to the output layer directly has shown to accelerate the training with similar accuracy results [23]. It is mentioned in their research that attaching the input features directly to output has the same flavor of Maximum Entropy features as in [40]. Feature collisions occurred

in [40] were due to the feature hashing technique adopted. Output labels in our case are less compared to the labels in language models, it is possible to have full connections between features and output in order to avoid the feature collision [23]. This can be used in future experiments to see if there is performance improvement or not as well. With accelerated training, the convergence might happen earlier and more runs can be tested with less time which would allow for more hyperparameter combination testing.

One of the main hurdles in this thesis was the unavailability of the properly labeled data. For speeding up the tagging in case of obtaining the manually tagged data one can use active learning. One of the research done in active learning for tagging the training data has been found useful where training instances may have multiple tags. This can be used in future experiments to obtain reliable labeling instances [13].

Factors like momentum setting and weight initialization strategies have the potential to improve training time and performance. These were not explored thoroughly in our experiments. Momentum is the size of the steps taken for getting towards minimum to try to avoid the local minima. This is for accelerating the training of models as well as improving the accuracy. One of the studies conducted on showing the importance of initialization and momentum [61] have shown that the initialization and momentum are crucial parameters for deep neural networks and RNNs. Previously it was considered that RNNs are difficult to train when using SGD with momentum. As our experiments have shown better performance with SGD, further research in improving the training with SGD can be done with momentum. Poorly initialized networks are difficult to train with momentum and well-initialized networks perform significantly worse when momentum is poorly tuned [61]. Even the research in hyperparameter optimization for LSTM has shown the performance changes based on random initialization. By keeping all the parameters constant if only the seed value is changed for random initialization then it has shown to give significant performance impacts on the test dataset [50]. This can be experimented for the currently achieved best parameters to find any performance impacts.

LSTM may suffer from the exploding gradient problem. Exploding gradient problem occurs when training deep neural networks using gradient descent by backpropagation which is the case with LSTM [47]. When large error gradients accumulate then the weights either underflow or overflow causing the model to become more unstable and impair effective learning. Strategies like gradient clipping and gradient normalization are used to avoid the exploding gradient problem. Gradient clipping clips the gradient's components element-wise if it goes beyond a defined threshold. Gradient normalization, on the other hand, has a better theoretical justification and it rescales the

gradient when the norm goes over a threshold. For sequence labeling tasks, gradient normalization as described by [48] improves significantly the performance. Performance improvement study in paper [50] showed that gradient clipping has not shown statistically significant performance improvements for sequence labeling tasks. Both these techniques can be tested in the future for observing if any performance improvements can be obtained by implementing these strategies for skill detection tasks.

The padding of the sentences to make all the input sizes to the network equal was not done in this thesis. Incorporating this would also enable to try out different mini batch sizes for future experiments. The optimal size of the mini batch has been shown to depend on the task [50], this optimal size can be found out for skill detection task. Along with this, different optimizers other than Adam, Nadam, SGD, and RMSProps can also be tested to find out the optimal optimizer for this task. In this thesis, we have experimented with the top better-performing optimizers found out for POS, Chunking, NER, Entity Recognition, and Event Detection as per [50]. Another strategy of making the performance better and making the model generalized is by adding the dropout layers. In our experiments only dropout in recurrent units was introduced with naive dropout strategy of static dropout rate. Variational dropout has shown to improve the performance for sequence labeling tasks [50] which can be experimented to find out its' effect on model learning for the skill detection task.

## Chapter 7

# Conclusions

Job recommendation is challenging and growing area of research in recent years. In this thesis, we have implemented a technique to help the job recommendation problem by identifying skills from job descriptions. Currently, there is no easy way to tag the job descriptions automatically with the skills present in it unless done with an exhaustive list of skills or manually labeling it. Methods used in this thesis help to automatically tag the descriptions with skills and have shown to perform better on unknown skill words. This sophisticated approach is an improvement over manual or direct keyword match since it takes into account the context and thus will avoid mislabeling of the skills when used in another context where it is not a skill. Tagging based on context will help in avoiding this ambiguity as well. The system built in this thesis addresses this exact problem and have shown satisfactory results to be used for real business needs.

We have shown the application of LSTM and BILSTM in automatically detecting skills from the job descriptions. The experiments achieved F1 score of **82** on job descriptions extracted from Stackshare platform. Total accuracy achieved on train skills was **94%** and on test skills **56%**. The results were in line with the similar research done in attribute finding from textual data [51] [74] [28]. Our results also indicated a significant difference in performance for with and without pre-trained word embeddings as input to the model. The results indicate clearly the use of pre-trained word embeddings trained using the Word2Vec model improved the detection of test skills and overall performance of the model. Similar results indicating performance improvements with the use of pre-trained word embeddings over randomly initialized word embeddings were shown in sequence labeling tasks [67]. Our results were also aligned with similar findings. These findings will be applicable in other sequence labeling tasks involving textual data. Using the context before and after the skill word has shown improved performance over using

only past dependencies. The experiments done in identifying more features have shown a promising approach for identifying the skills from job descriptions using BILSTM method which uses pre-trained word embeddings, POS tags and capitalization of the first letter of the word.

# Bibliography

- [1] AL-OTAIBI, S. T., AND YKHLEF, M. A survey of job recommender systems. *International Journal of Physical Sciences* 7, 29 (2012), 5127–5142.
- [2] BENGIO, Y., DUCHARME, R., VINCENT, P., AND JAUVIN, C. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.
- [3] BENGIO, Y., AND LECUN, Y., Eds. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015).
- [4] BOWMAN, S. R., POTTS, C., AND MANNING, C. D. Recursive neural networks for learning logical semantics. *CoRR abs/1406.1827* (2014).
- [5] CHENG, G., PEDDINTI, V., POVEY, D., MANOHAR, V., KHUDANPUR, S., AND YAN, Y. An exploration of dropout with lstms. In *Interspeech* (2017), pp. 1586–1590.
- [6] CHIU, J., AND NICHOLS, E. Sequential labeling with bidirectional lstm-cnns. In *Proc. International Conf. of Japanese Association for NLP* (2016), pp. 937–940.
- [7] CHIU, J. P., AND NICHOLS, E. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics* 4 (2016), 357–370.
- [8] CHO, K., VAN MERRIËNBOER, B., BAHDANAU, D., AND BENGIO, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).
- [9] CHO, K., VAN MERRIENBOER, B., BAHDANAU, D., AND BENGIO, Y. On the properties of neural machine translation: Encoder-decoder approaches. *CoRR abs/1409.1259* (2014).

- [10] CHUNG, J., GULCEHRE, C., CHO, K., AND BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
- [11] COLLOBERT, R., WESTON, J., BOTTOU, L., KARLEN, M., KAVUKCUOGLU, K., AND KUKSA, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* 12 (Nov. 2011), 2493–2537.
- [12] COLLOBERT, R., WESTON, J., BOTTOU, L., KARLEN, M., KAVUKCUOGLU, K., AND KUKSA, P. Natural language processing (almost) from scratch. *Journal of machine learning research* 12, Aug (2011), 2493–2537.
- [13] CULOTTA, A., AND MCCALLUM, A. Reducing labeling effort for structured prediction tasks. In *AAAI* (2005), vol. 5, pp. 746–751.
- [14] DENG, L., AND YU, D. *Deep Learning: Methods and Applications*, vol. 7 ed. Foundations and Trends® in Signal Processing, 2014.
- [15] DEY, R., AND SALEMT, F. M. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (Aug 2017), pp. 1597–1600.
- [16] GOLDBERG, Y., AND LEVY, O. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *CoRR abs/1402.3722* (2014).
- [17] GOLLER, C., AND KUCHLER, A. Learning task-dependent distributed representations by backpropagation through structure. In *Proceedings of International Conference on Neural Networks (ICNN’96)* (1996), vol. 1, IEEE, pp. 347–352.
- [18] GRAVES, A., MOHAMED, A.-R., AND HINTON, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (2013), IEEE, pp. 6645–6649.
- [19] GRAVES, A. . J. S. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks* 18(5-6), 602-610 (2005).
- [20] HAGAN, M. T., DEMUTH, H. B., BEALE, M. H., AND DE JESÚS, O. *Neural network design*, vol. 20. Pws Pub. Boston, 1996.



- [21] HEMATI, W., AND MEHLER, A. Lstmvoter: Chemical named entity recognition using a conglomerate of sequence labeling tools. *Journal of Cheminformatics* 11 (12 2019).
- [22] HOCHREITER, S., AND SCHMIDHUBER, J. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
- [23] HUANG, Z., XU, W., AND YU, K. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991* (2015).
- [24] IYYER, M., BOYD-GRABER, J., CLAUDINO, L., SOCHER, R., AND DAUMÉ III, H. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2014), pp. 633–644.
- [25] JAGANNATHA, A. N., . Y. H. Bidirectional rnn for medical event detection in electronic health records. *Association for Computational Linguistics. North American Chapter. Meeting (Vol. 2016, p. 473)* (2016).
- [26] JAGANNATHA, A. N., AND YU, H. Structured prediction models for rnn based sequence labeling in clinical text. In *Proceedings of the conference on empirical methods in natural language processing. conference on empirical methods in natural language processing* (2016), vol. 2016, NIH Public Access, p. 856.
- [27] JUNG, A. A gentle introduction to supervised machine learning. *CoRR abs/1805.05052* (2018).
- [28] KATSUTA, AKIHIRO, H. A. H. S. A. S. H. K. T. Y. U., AND MATSUMOTO, Y. Infomation extraction from english & japanese résumé with neural sequence labelling methods. In *The Association for Natural Language Processing* (2018), Nagaoka University of Technology.
- [29] KUMARAN, V. S., AND SANKAR, A. Towards an automated system for intelligent screening of candidates for recruitment using ontology mapping (expert). *International Journal of Metadata, Semantics and Ontologies* 8, 1 (2013), 56–64.
- [30] L. AUBERT, X. An overview of decoding techniques for large vocabulary continuous speech recognition. *Computer Speech & Language* 16 (01 2002), 89–114.
- [31] LAMPLE, G., BALLESTEROS, M., SUBRAMANIAN, S., KAWAKAMI, K., AND DYER, C. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360* (2016).

- [32] LAUMER, S., AND ECKHARDT, A. Help to find the needle in a haystack: integrating recommender systems in an it supported staff recruitment system. In *Proceedings of the special interest group on management information system's 47th annual conference on Computer personnel research* (2009), ACM, pp. 7–12.
- [33] LING.UPENN.EDU. Alphabetical list of part-of-speech tags used in the penn treebank project. webpage, 2019. [https://www.ling.upenn.edu/courses/Fall\\_2003/ling001/penn\\_treebank\\_pos.html](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html).
- [34] LIU, P., JOTY, S. R., AND MENG, H. M. Fine-grained opinion mining with recurrent neural networks and word embeddings. In *EMNLP* (2015).
- [35] MALINOWSKI, J., KEIM, T., WENDT, O., AND WEITZEL, T. Matching people and jobs: A bilateral recommendation approach. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06)* (2006), vol. 6, IEEE, pp. 137c–137c.
- [36] MEFTAH, S., AND SEMMAR, N. A neural network model for part-of-speech tagging of social media texts. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)* (2018).
- [37] MERITY, S., KESKAR, N. S., AND SOCHER, R. Regularizing and optimizing LSTM language models. *CoRR abs/1708.02182* (2017).
- [38] MESNIL, G., HE, X., DENG, L., AND BENGIO, Y. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech* (2013), pp. 3771–3775.
- [39] MIKOLOV, T., CHEN, K., CORRADO, G., AND DEAN, J. Efficient estimation of word representations in vector space, 2013.
- [40] MIKOLOV, T., DEORAS, A., POVEY, D., BURGET, L., AND ČERNOCKÝ, J. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition Understanding* (Dec 2011), pp. 196–201.
- [41] MIKOLOV, T., KARAFIÁT, M., BURGET, L., ČERNOCKÝ, J., AND KHUDANPUR, S. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association* (2010).

- [42] MIKOLOV, T., KOMBRINK, S., DEORAS, A., BURGET, L., AND CERNOCKY, J. Rnnlm-recurrent neural network language modeling toolkit. In *Proc. of the 2011 ASRU Workshop* (2011), pp. 196–201.
- [43] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G., AND DEAN, J. Distributed representations of words and phrases and their compositionality. *CoRR abs/1310.4546* (2013).
- [44] MIKOLOV, T., SUTSKEVER, I., CHEN, K., CORRADO, G. S., AND DEAN, J. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (2013), pp. 3111–3119.
- [45] MIYATO, T., DAI, A. M., AND GOODFELLOW, I. Adversarial Training Methods for Semi-Supervised Text Classification. *arXiv e-prints* (May 2016), arXiv:1605.07725.
- [46] OUALIL, Y., SINGH, M., GREENBERG, C., AND KLAOW, D. Long-short range context neural networks for language modeling. *arXiv preprint arXiv:1708.06555* (2017).
- [47] PASCANU, R., MIKOLOV, T., AND BENGIO, Y. Understanding the exploding gradient problem. *CoRR, abs/1211.5063 2* (2012).
- [48] PASCANU, R., MIKOLOV, T., AND BENGIO, Y. On the difficulty of training recurrent neural networks. In *International conference on machine learning* (2013), pp. 1310–1318.
- [49] PAZZANI, M. J., AND BILLSUS, D. Content-based recommendation systems. In *The adaptive web*. Springer, 2007, pp. 325–341.
- [50] REIMERS, N., AND GUREVYCH, I. Optimal hyperparameters for deep lstm-networks for sequence labeling tasks. *arXiv preprint arXiv:1707.06799* (2017).
- [51] SAWANT, U., AND GABALE, V. Product discovery from e-commerce listings via deep text parsing. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data* (New York, NY, USA, 2018), CoDS-COMAD ’18, ACM, pp. 98–107.
- [52] SCHMITT, T., CAILLOU, P., AND SEBAG, M. Matching jobs and resumes: a deep collaborative filtering task. In *GCAI 2016-2nd Global Conference on Artificial Intelligence* (2016), vol. 41.

- [53] SINGH, M., SMIT, P., VIRPIOJA, S., KURIMO, M., ET AL. First-pass decoding with n-gram approximation of rnnlm: The problem of rare words. In *Workshop on Machine Learning in Speech and Language Processing* (2018).
- [54] SNOEK, J., LAROCHELLE, H., AND ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.
- [55] SOCHER, R., HUANG, E. H., PENNIN, J., MANNING, C. D., AND NG, A. Y. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2011, pp. 801–809.
- [56] SOCHER, R., LIN, C. C., MANNING, C., AND NG, A. Y. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (2011), pp. 129–136.
- [57] SOCHER, R., PERELYGIN, A., WU, J., CHUANG, J., MANNING, C. D., NG, A., AND POTTS, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing* (2013), pp. 1631–1642.
- [58] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
- [59] SURDEANU, M. Overview of the tac2013 knowledge base population evaluation: English slot filling and temporal slot filling. In *TAC* (2013).
- [60] SUTSKEVER, I., MARTENS, J., DAHL, G., AND HINTON, G. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, Georgia, USA, 17–19 Jun 2013), S. Dasgupta and D. McAllester, Eds., vol. 28 of *Proceedings of Machine Learning Research*, PMLR, pp. 1139–1147.
- [61] SUTSKEVER, I., MARTENS, J., DAHL, G., AND HINTON, G. On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (2013), pp. 1139–1147.

- [62] THIAGARAJAN, R., MANJUNATH, G., AND STUMPTNER, M. *Finding experts by semantic matching of user profiles*. PhD thesis, CEUR-WS, 2008.
- [63] TIELEMAN, T., AND HINTON, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA* (2012), 26–31.
- [64] TILK, O., AND ALUMÄE, T. Lstm for punctuation restoration in speech transcripts. In *Sixteenth annual conference of the international speech communication association* (2015).
- [65] TILK, O., AND ALUMÄE, T. Bidirectional recurrent neural network with attention mechanism for punctuation restoration. In *Interspeech* (2016), pp. 3047–3051.
- [66] WANG, P., Q. Y. S. F. K. H. L. . Z. H. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168 (2015)* (2015).
- [67] XIN, Y., HART, E., MAHAJAN, V., AND RUVINI, J.-D. Learning better internal structure of words for sequence labeling. *arXiv preprint arXiv:1810.12443* (2018).
- [68] XU, K., X. L. . Y. K. Investigating lstm for punctuation prediction. *10th International Symposium on Chinese Spoken Language Processing (ISCSLP) (pp. 1-5)* (2016).
- [69] YANG, J., AND ZHANG, Y. Ncrf++: An open-source neural sequence labeling toolkit. *arXiv preprint arXiv:1806.05626* (2018).
- [70] YAO, K., P. B. Z. Y. Y. D. Z. G. . S. Y. Spoken language understanding using long short-term memory neural networks. *IEEE Spoken Language Technology Workshop (SLT) (pp. 189-194)* (2014, December).
- [71] YIN, W., KANN, K., YU, M., AND SCHÜTZE, H. Comparative study of CNN and RNN for natural language processing. *CoRR abs/1702.01923* (2017).
- [72] ZHANG, L., WANG, S., AND LIU, B. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8, 4 (2018), e1253.
- [73] ZHANG, Q., YANG, L. T., CHEN, Z., AND LI, P. A survey on deep learning for big data. *Information Fusion* 42 (2018), 146 – 157.

- [74] ZHAO, M., JAVED, F., JACOB, F., AND MCNAIR, M. Skill: A system for skill identification and normalization. In *Twenty-Seventh IAAI Conference* (2015).
- [75] ZHAO, Z., LIU, T., LI, S., LI, B., AND DU, X. Ngram2vec: Learning improved word representations from ngram co-occurrence statistics. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (Copenhagen, Denmark, sep 2017), Association for Computational Linguistics, pp. 244–253.